University of Alberta

Eikon: A Tool for Geometric Coding of Large Multivariate Datasets

by

Paul A. Ferry ©

A thesis submitted to the Faculty of Graduate Studies and Research in partial fulfillment of the requirements for the degree of **Doctor of Philosophy**.

Department of Computing Science

Edmonton, Alberta
Fall 2002

National Library
of Canada

Acquisitions and
Bibliographic Services

395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque nationale
du Canada

Acquisitions et
services bibliographiques

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence

Our file Notre référence

The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

L'auteur conserve la propriété du droit d'auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

0-612-81188-3

Canada

University of Alberta

Library Release Form

Name of Author: **Paul A. Ferry**

Title of Thesis: **Eikon: A Tool for Geometric Coding of Large Multivariate Datasets**

Degree: Doctor of Philosophy

Year this Degree Granted: 2002

Permission is hereby granted to the University of Alberta Library to reproduce single copies of this thesis and to lend or sell such copies for private, scholarly or scientific research purposes only.

The author reserves all other publication and other rights in association with the copyright in the thesis, and except as hereinbefore provided, neither the thesis nor any substantial portion thereof may be printed or otherwise reproduced in any material form whatever without the author's prior written permission.

Paul Ferry

Paul A. Ferry
Roosky
Newtowncunningham
Co. Donegal
Ireland

Date: Sept 11 2002

The Bellman himself they all praised to the skies–
Such a carriage, such ease and such grace!
Such solemnity, too! One could see he was wise,
The moment one looked in his face!

He had brought a large map representing the sea,
Without the least vestige of land:
And the crew were much pleased when they found it to be
A map they could all understand.

"What's the good of Mercator's North Poles and Equators,
Tropics, Zones and Meridian lines?"
So the Bellman would cry: and the crew would reply,
"They are merely conventional signs!

"Other maps are such shapes, with their islands and capes!
But we've got our brave captain to thank
(So the crew would protest) "that he's brought us the best–
perfect and absolute blank!"

–Lewis Carroll, The Hunting of the Snark.

University of Alberta

Faculty of Graduate Studies and Research

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies and Research for acceptance, a thesis entitled **Eikon: A Tool for Geometric Coding of Large Multivariate Datasets** submitted by **Paul A. Ferry** in partial fulfillment of the requirements for the degree of **Doctor of Philosophy**.

Dr. John Buchanan

Dr. Pierre Boulanger

Dr. Jim Hoover

Roger W Toogood

Dr. Rodger Toogood

Dr. Osmar Zaiane

Dr. Sean Gouglas

Dr. Alex Pang

Date: July 02 2002

To Sarda
vspace0.33inWill you marry me?

Yes

No

Circle as appropriate

Abstract

With the increase in the amount of data being generated by today's methods of simulation, experimentation, and direct measurement, there has been a growing interest in the use of graphical representations of data to aid in its exploration, analysis, and interpretation

One such graphical representation, geometric coding, uses parameterized geometric objects, called glyphs or icons, to encode the data by varying the geometric properties of each glyph with respect to the underlying data. Geometric coding allows integrated images of multiple variables to be constructed, however, as the number of data points displayed increases, the amount of visible variation generated per glyph decreases, potentially obscuring the visibility of interesting structures and patterns in the data.

In this thesis, I present a new system, called Eikon, that extends the use of geometric coding methods to large multivariate datasets. Eikon is based on the determination of interesting features in the data at multiple scales and the distribution and display of a variable number of glyphs based on the location and size of these features. Using this technique Eikon generates images where important data features are displayed through variation in glyph density at one scale and where elementary details of data points are displayed via geometric variation in individual glyphs at another scale.

The generation and exploration of data views in Eikon is facilitated by an interactive visual programming environment. This allows images of the data to be quickly generated that focus on different aspects of the data. It also allows different images of the data to be generated using different glyphs or different bindings between data variables and glyph parameters. Additionally Eikon allows the number of glyphs displayed to be adjusted supporting various levels of detail and providing the necessary

interactivity for visually exploring the data at multiple scales.

The thesis is concluded by describing the results of a visualization case study in which Eikon is shown to be a useful tool in the visualization of large multivariate data. The case study involves visualizing the results from a geophysical fluid dynamics simulation and demonstrates how Eikon can produce images similar to existing geometric coding methods, and also how several new images can be generated that focus on different user selected important attributes of the dataset. The case study shows how global overviews can be created and how various levels of detail views that show both global and more elementary information can be produced. At each step the utility of Eikon is illustrated through the advantages and disadvantages described by the case study participants.

Acknowledgements

This thesis would have been impossible without the input from a great many people. First and foremost my supervisors Dr. John Buchanan and Dr. Pierre Boulanger. Special thanks also go to Dr. Mark Green for his supervision while at the University of Alberta. Thanks must also be extended to the other members of my supervisory committee, namely Dr. Jim Hoover and Dr. Roger Toogood, and to the remaining members of the examination committee, Dr. Osmar Zaine, Dr. Sean Gouglas, and last but by no means least Dr. Alex Pang who traveled to Edmonton from Santa Cruz, California for the final examination.

Special thanks are also extended to Edith Drummond, who from the very beginning, has provided immeasurable assistance to me in completing my studies and research at the University of Alberta. A large debt of gratitude is also owed to the many and varied students of the Computer Graphics Research Lab since 1994. Of special note are Dr. Chris Shaw, Dr. Oleg Veryovka, Lisa Streit, Sean Harrison, Dr. Mario Costa Sousa, and Pablo Figueroa who where all instrumental in the formulation and development of the ideas behind this thesis. Thanks are also given to Maurice Black, Tara Tapics, and Tracie Scott for invaluable proof reading and editorial input.

I must also thank the many friends that I've met along the way. Namely Edmund Gill, Dwayne Walker, Kim Musselwhite, Greg Hornby, John Doucette, Lee Hyun Jeong, Chris Thompson, Robyn Taylor, Cindy Want, and Anita Petrinjak. One couldn't ask to know better people. Last and absolutely by no means least I am eternally grateful to Sarda Sahney for being with me, and perhaps, more importantly for staying with me in the years to come.

Contents

1 Introduction **1**

1.1 Research Context and Motivation 2

1.2 Objectives of the Research 3

1.3 Overview of the Current Work 6

1.3.1 Abstraction Functions 6

1.3.2 Multi-level Importance Representation 7

1.3.3 Glyph Distribution 7

1.4 Evaluation 7

1.5 Thesis Structure 8

2 Fundamentals of Multivariate Data Display **11**

2.1 Introduction 11

2.2 Image Construction 12

2.2.1 Properties of the Data 12

2.2.2 Properties of the Image 13

2.2.3 Data Display Models 16

2.3 Image Legibility 16

2.3.1 Data Context and Encoding 17

2.3.2 Viewer Requirements 17

2.3.3 Levels of Reading 18

2.3.4 Levels of Organization 19

2.4 Conclusion 20

3 Visual Coding **21**

3.1 Introduction 21

3.2 Color Coding 25

3.2.1 Color Utilization for Data Display 28

3.2.2 Color Coding Legibility 28

3.2.3 Future Work 33

3.3 Geometric Coding 34

3.3.1 Geometric Utilization for Data Display 34

3.3.2 Geometric Coding Legibility 39

3.3.3 Future Work 41

3.4 Texture Coding 43

3.4.1 The Dimensions of Texture 43

3.4.2 Texture Utilization for Data Display . . . 44
3.4.3 Texture Legibility . . . 46
3.4.4 Future Work . . . 47
3.5 Summary and Conclusions . . . 48
3.5.1 Conclusion . . . 49

4 Spatial Utilization for Data Display **51**
4.1 Introduction . . . 51
4.2 Scale-Space Methods . . . 52
4.2.1 Pan and Zoom . . . 52
4.2.2 Spatial Deformation . . . 54
4.2.3 Scale-Space Diagrams . . . 56
4.3 Data Abstraction . . . 58
4.3.1 Data Probing . . . 59
4.3.2 Autonomous Probes . . . 59
4.3.3 Conditional Display . . . 60
4.3.4 Feature Extraction . . . 60
4.4 Conclusion . . . 62
4.4.1 The Eikon System . . . 63

5 ETK: The Eikon Toolkit **67**
5.1 Introduction . . . 67
5.2 System Overview . . . 68
5.2.1 Transformation and Abstraction . . . 68
5.2.2 Multi-scale Importance and Distribution . . . 70
5.2.3 Data display . . . 74
5.3 System Design and Implementation . . . 75
5.3.1 Data Representation . . . 76
5.3.2 Dataset Interface . . . 79
5.3.3 Process Representation . . . 80
5.4 Execution and Control . . . 89
5.4.1 Centralized Control . . . 89
5.4.2 Distributed Control . . . 90
5.4.3 Execution and Control in Eikon . . . 90
5.5 Example Code . . . 94
5.6 Summary . . . 97

6 The Visual Programming Environment **98**
6.1 Introduction . . . 98
6.2 System Overview . . . 100
6.3 Visualization Networks . . . 103
6.3.1 Creating Networks . . . 103
6.3.2 Modification and Rendering . . . 105
6.3.3 Reading, Writing, and Code Generation . . . 106
6.4 Tools, Properties, and Configuration . . . 107

6.4.1 Tools 107
6.4.2 Properties 109
6.4.3 Data Fields 111
6.5 Summary 113

7 Case Study - Introduction **115**
7.1 Introduction 115
7.2 Density-driven Benthic flows 116
7.3 Model Overview 117
7.3.1 Generated Data 119
7.3.2 Model Verification 119
7.4 Case Study Objective 119

8 Case Study - Elementary Views **121**
8.1 Elementary Views in Eikon 122
8.2 Local Geometry and Texture 129
8.3 Color 132
8.4 Multi-glyphs and Multi-layers 133
8.5 Discussion 136
8.5.1 Advantages 136
8.5.2 Disadvantages 140

9 Case Study - Global Views **142**
9.1 Global Overviews in Eikon 143
9.1.1 Importance Functions 144
9.1.2 Abstraction Functions 148
9.1.3 Velocity and Vorticity Views 153
9.1.4 Multi-layer Views 157
9.2 Level of Detail 157
9.3 Discussion 163
9.3.1 Advantages 164
9.3.2 Disadvantages 166

10 Case Study - Intermediate Views **167**
10.1 Intermediate Views in Eikon 168
10.2 Importance Functions and Multiple Variables 171
10.3 Multi-glyph and Multilayer Views 176
10.3.1 Multilayer Views 176
10.4 Discussion 179
10.4.1 Advantages 179
10.4.2 Disadvantages 181

11 Case Study - Conclusion **183**
11.1 Benefits 183
11.1.1 Global Overviews 184

    - 11.1.2 Intermediate Views 184
    - 11.1.3 Elementary Views 185
  - 11.2 Shortcomings 186
    - 11.2.1 Discrete Displays 186
    - 11.2.2 Color Variation 186
    - 11.2.3 Local Detail 187
    - 11.2.4 Other methods 187
  - 11.3 Conclusion 188

12 Conclusion **189**

  - 12.1 Abstraction and Importance 190
  - 12.2 Distribution and Data Display 191
  - 12.3 Benefits of the current system 191
    - 12.3.1 Global Overviews 191
    - 12.3.2 Intermediate and Elementary Views 192
    - 12.3.3 Visual Programming Environment 193
  - 12.4 Limitations and Future Considerations 194

List of Tables

8.1 The bindings for the face glyph (elementary view). 123
8.2 The bindings assigned to the profile glyph(elementary view). 128
8.3 The binding assigned to the line glyph (elementary view). 129
8.4 The binding assigned to the stick figure glyph (elementary view). . . . 129

9.1 The binding assigned to the star glyph (global view). 145
9.2 The bindings assigned to the profile glyph (global view). 147
9.3 The bindings assigned to the star glyph (global view). 153
9.4 The bindings assigned to the profile glyph (global view). 154
9.5 Frame rates for various levels of detail. 162

10.1 The bindings assigned to the profile glyph (intermediate view). 168
10.2 The bindings assigned to the stick figure glyph (intermediate view). . 169
10.3 The binding assigned to the line glyph (intermediate view. 171
10.4 The binding assigned to line glyph (height and pressure view). 173
10.5 The binding assigned to the line glyph (pressure and vorticity view). . 173
10.6 The bindings assigned to the profile glyph. 178

List of Figures

2.1 Properties of visual variables allowed by a static two-dimensional image. 14

3.1 The visualization pipeline. 22
3.2 The CIE chromaticity diagram and color gamut comparisons. 26
3.3 Examples of simultaneous contrast effecting brightness and saturation. 30
3.4 Examples of color schemes following Trumbo's Principles III and IV. 32
3.5 Robertson and O'Callaghans double cone scheme. 33
3.6 Examples of several different glyphs as presented in the literature. . 35
3.7 Local optical illusions due to geometry. 42
3.8 Global optical illusions due to geometry and intensity. 43

4.1 The perspective wall and document lens. 54
4.2 Panning and zooming in scale-space. 57
4.3 Spatial deformation in scale-space. 57

5.1 The system architecture of Eikon. 69
5.2 Computing the multi-scale importance representations. 71
5.3 Computing importance weights for glyph distribution. 73
5.4 The etkDataSet and etkDataType class hierarchy. 77
5.5 Process representation in ETK. 81
5.6 The etkDataSource class hierarchy. 82
5.7 The etkDataFilter class hierarchy. 83
5.8 The etkDataSink class hierarchy. 85
5.9 The glyphs implemented in Eikon. 87
5.10 The Pull/Push pattern through a network in ETK. 92
5.11 The function call and execution pattern of a Pull/Push cycle. 93

6.1 The system architecture of the visual editor. 100
6.2 The visual editor layout. 101
6.3 An example network in the visual editor. 104
6.4 Visualization network representation in the visual editor. 105
6.5 Tools and their properties. 108
6.6 The etkTool class hierarchy. 109
6.7 The etkProperty class hierarchy. 110
6.8 The data fields defined in the visual editor. 112

7.1 The ocean model. 118

8.1 Basic visualization network. 123
8.2 An example of a glyph configuration dialog. 124
8.3 Zoomed elementary view with the face glyph. 125
8.4 Un-zoomed views using the face glyph. 126
8.5 Elementary view using the profile glyph. 128
8.6 A larger scale elementary view using the line glyph. 130
8.7 Four elementary views using different bindings for the stick figure glyph. 131
8.8 The color sequence editor. 133
8.9 Combining color with (stick figure) glyph variation. 134
8.10 Multi-layered. multi-glyph image using the profile and line glyph . . 135
8.11 Colored multi-layered, multi-glyph image using the profile and line glyph. 137
8.12 Overlapping multi-layered, multi-glyph image using the profile and line glyph. 138

9.1 A visualization network using importance. 143
9.2 The color scale used in the height variation global view 145
9.3 Height variation overview. 146
9.4 The color sequenced used for pressure magnitude (global view.) . . . 148
9.5 A global view of pressure magnitude. 149
9.6 Examples of glyph distribution (height) and color in global overviews. 150
9.7 Examples of glyph distribution (pressure) and color in global overviews. 151
9.8 A network using abstraction and importance. 154
9.9 Velocity Magnitude. 155
9.10 Vorticity Variance. 156
9.11 Velocity and vorticity abstractions using color. 158
9.12 Multi-layered image using two abstraction functions. 159
9.13 Reducing global level of detail. 161

10.1 Height variation with a profile glyph (intermediate view). 169
10.2 Pressure magnitude using the stick figure glyph and color. 170
10.3 Pressure magnitude using line glyph and color (intermediate view). . 172
10.4 Pressure and height combined. 174
10.5 Pressure and vorticity combined. 175
10.6 Zoomed intermediate views. 177
10.7 Multiple glyphs using a single importance function. 178
10.8 Multi-layered, multi-glyph image using two different distributions of the profile and line glyph. 180

Chapter 1

Introduction

> *"If we trace out what we behold and experience through the language of logic we are doing science; if we show it in forms whose interrelationships are not accessible to our conscious thought but are intuitively recognized as meaningful, we are doing art."*
>
> *Albert Einstein*

The usefulness of graphics as a means of presenting information has been know for a very long time. The earliest use of graphics consisted of simple maps drawn on clay tablets, dating back as far as 3,000 BC. These maps corresponded to simple scaled representations of physical space and it wasn't until 200 years ago that people such as Lambert [lamb79] and Playfair [play86] realized that the two dimensions of the page could also be utilized to encode other more abstract variables[1].

More recently, in 1987 a special report to the National Science Foundation [mcco87] reiterated the importance of visualization in scientific computing where modern sources of data have been called such "fire hoses of information." The goal of the research described in this thesis is to provide a new tool for generating images to aid in the analysis and interpretation of one class of this data: large and multivariate data defined on a two-dimensional grid.

---

[1] We are deliberately excluding art work throughout the ages in this context.

1.1 Research Context and Motivation

Scientists are generating gigabytes of data daily. These datasets stem from wide and varied sources, such as computational fluid dynamics, thermodynamics, earth resource and atmospheric science, oceanographic research, and astronomical and geological studies. But the datasets share a common problem: the ease with which the data can be generated far outstrips the ease with which it can be analyzed. This problem has been called "the information-without-interpretation dilemma" [mcco87] and has led to large quantities of data being collected and stored without any form of further observation.

In many cases there are no tools suitable for the analysis or exploration of these large datasets. As a result scientific visualization [niel90, gall95] has emerged as a relatively new field with the goal of providing such tools. Visual representation of data allow scientists to readily form mental images of the information they contain, providing leverage for their analysis and interpretation. This explains the increasing emphasis being place on scientific visualization today.

It is important to note, however, that visualization offers interpretative leverage *only if* the necessary information a dataset contains is rendered visible in the generated image [clev94]. The research described in this dissertation is concerned with this question in regards to large two-dimensional multivariate data display. I characterize this problem as the need to display a dataset with dimensions of at least 1,024 by 1,024 and where each data point is defined by two or more variables. Furthermore I am interested in rendering images of these datasets where the image has a resolution of 1,024 by 1,024 or less. Generating images of such data presents two problems:

1. The overwhelming amount of data to be displayed. The display of all data points contained in the dataset produces images in which everything is displayed, but in which very little is actually visible. In [clev94], Cleveland refers to this as the problem of detection. He indicates that "detection is in a sense the most fundamental perceptual issue" (page 239) meaning that before action can be taken based on the data the needed aspects of the data must first be rendered visible. Cleveland goes on to say that in some cases "the issue of detection...is so obvious that it is conceptually trivial, although it is not at all trivial to deal

with the problem in practice." Furthermore displaying all the data can generate a computation and rendering load which strains the capabilities of the display hardware.

2. The limited visual stimuli which can be used to display the data.

   The value of each variable that defines a data point must be represented by variation in the visual properties of the image. Many data variables may be defined, but a static image offers only eight visual stimuli: x and y position, size, shape, orientation, color, intensity, and texture [tuft83, tuft91, bert83].

Geometric Coding

A popular way of displaying multivariate data has been to use geometric coding. Geometric coding encodes data by varying the geometric properties of parameterized geometric objects, called glyphs or icons[2], with respect to the underlying data. Using this technique, characteristics of individual variables, and their relationships to other variables, can be perceived from individual or small groups of glyphs. With careful crafting, it is also possible to generate a texture by displaying many small but discernible glyphs in the image, enabling global patterns and structures to be seen.

Although geometric coding allows the generation of integrated images of many variables they do not scale well with dataset size. For large datasets, containing millions of data points, this cost can be high. Only a small area is available to display each individual glyph and as a result the amount of visible variation due to each glyph is reduced producing images where the visibility of important structures and patterns in the data can be obscured.

1.2 Objectives of the Research

The goal of this research is to extend the use of geometric coding techniques to large multivariate datasets. To do this I develop a system that takes advantage of two observations from previous work:

1. Displaying all the data in a single dataset can hinder analysis and interpretation.

---

[2] There is no universally accepted terminology and these two terms are often used interchangeably.

Displaying large amounts of data requires a great deal of rendering power and places equal emphasis on all of it. Reducing the data to important or interesting features and displaying only these can reduce the amount of data to be rendered, focus attention on its important characteristics, and provide more space for displaying its details.

2. The information encoding potential of the spatial dimension is under utilized

   The spatial dimension is regarded as one of the more flexible visual dimensions for data display [bert83, clev94] and therefore exploiting this flexibility would seem like a worthwhile endeavor.

The concept of reducing data to its important features and rendering these features forms the basic idea behind the tool presented here. This tool is based on a technique that detects features of interest at multiple scales, and distributes a user defined number of glyphs over the image based on these features. The objective of this approach is to support the visual variation of individual or small groups of glyphs when the scale permits and to use variation in glyph density at other scales. Using appropriate glyphs[3], variation in individual glyphs can show elementary variation on a per data point basis while variation in glyph density can show global patterns and structures in the data. I call the system *Eikon* from the Greek word *eikon* meaning *image* and from which the English word *icon* is derived. This captures the notion of using individual glyphs or *icons* to represent individual data points and also for all glyphs together to form an *image* of the data they encode.

Contribution

Eikon contributes to the display of large multivariate datasets using geometric coding in a number of ways.

1. Emulation of existing techniques

   Eikon can produce results similar to existing techniques by using a distribution which corresponds to their placement policy for individual glyphs. For example many techniques place a glyph at each data point. This behavior is replicated by uniformly distributing glyphs over the image.

---

[3] This thesis is not concerned about the various capabilities of individual glyphs but in the production of a tool that supports the use of a wide variety of different glyphs

2. Overviews of large datasets

   When millions of data points must be considered, little or no variation is possible on an individual glyph bases. At this scale, using spatial distribution of glyphs, Eikon creates an image where variation in the glyph density renders important characteristics of the data visible.

3. Enhanced intermediate views of large datasets.

   The spatial distribution process, combined with reduced numbers of glyphs to display the data, allows intermediate level views to be produced in which global information is visible through glyph density, and elementary information is visible through local glyph variation. Furthermore layered or composite images can be generated by basing the distribution on two or more interesting features from the data.

In generating the different views of the data Eikon also contributes in the following ways:

1. Control over level of detail

   By distributing a variable number of glyphs, using a multi-scale representation of important features, Eikon can generate different level of detail views of the data. When many data points need to be displayed reducing the number of glyphs reduces the rendering load and improves the interactivity needed for data exploration. The multi-scale importance representation ensures as the number of glyphs are reduced the most important features remain visible. The distribution structure also ensures that, as the number of glyphs are reduce, a near continuous reduction in detail is presented rather than discontinuous jumps associated with many other techniques.

2. Less dependence on geometric properties of icons

Eikon is less dependent on specific geometric properties of individual glyphs. If information concerning the data is not represented in individual glyphs, important information is still visible through the spatial distribution of glyphs in the image. Eikon is not, however, independent of the geometry of each glyph. When elementary details of data points are important the effectiveness of the display is still dependent on the geometry of each glyph and its binding to data variables. This problem lies outside the scope of this thesis.

3. Multiple views of the data

   Eikon supports visual exploration of the data by allowing multiple views of the data to be generated quickly. Views of the data are generated by using a visual programming environment to construct visualization networks and to quickly change the data that is being displayed, the data characteristics that are being focused on, and the glyph that is being used to render the data. The programming environment also allows the parameters of the visualization to be adjusted such as the binding between glyph parameters and data variables and the number of glyphs used to display the data.

1.3 Overview of the Current Work

Eikon controls the spatial distribution of glyphs by rendering the data using a three-step methodology: (1) the application of *abstraction functions* to determine interesting features in the data; (2) the construction of a *multi-scale importance* representation of the abstraction; and (3) the *distribution* of glyphs based on this importance representation.

1.3.1 Abstraction Functions

Abstraction functions are used to compute features of interest from the data that is to be displayed. Abstraction functions, typically, derive information from the original data that it currently does not explicitly represent. In this sense abstraction is used as a method of transforming the data and representing the information it contains in different ways.

1.3.2 Multi-level Importance Representation

An importance representation of the data is computed by applying an *importance function* to the abstracted or original data. The importance function returns a single scalar value which represents some characteristic, such as variance or magnitude, of the data. In a manner similar to MIP-mapping [will83] a multi-level importance pyramid is computed by recursively applying an averaging filter to the computed importance representation.

1.3.3 Glyph Distribution

Glyphs are distributed over the image based on the multi-scale importance pyramid. The apex of the pyramid corresponds to an average of the importance values for the entire dataset. The next level contains four importance values corresponding to the four quadrants of the dataset. Each of these quadrants, in turn, is divided into four regions whose importance values are represented by four importance values in the next lower level. Glyphs are distributed, in a recursive manner, starting at the apex and determining the percentage of glyphs that should be assigned to the four different regions at the lower levels. The percentage of glyphs to distributed to a particular region is computed by summing the four importance values for each of the four regions at the lower level and determining the percentage each region contributes to this sum. This percentage value indicates the percentage of available glyphs that will be distributed over the individual regions. The distribution process continues down through the importance pyramid until the lowest level is reached or no glyphs remain to be distributed. Given the applied abstraction function, importance function and number of glyphs, the region in the latter case is deemed unimportant. In the former case, up to four glyphs are placed, starting with the data point with the largest importance value. This distribution process results in an image that correlates importance with the spatial density of glyphs.

1.4 Evaluation

The objective of this research is to produce a visualization tool that allows geometric coding to be used for the display and exploration of large two-dimensional multivariate data. I evaluate this research with a case study that describes how Eikon was used in the visualization of a simulation dataset produced by the fluid dynamics research group in the Department of Mathematics at the University of Alberta. The case study demonstrates how several abstraction and importance functions can be used in conjunction with different glyphs to produce numerous images of large multivariate datasets. It shows how existing results can be emulated, how global overviews can be constructed and how level of detail views showing both global and elementary information on a per data point basis can be generated. At each step I describe how the visual programming environment is used to construct the visualizations and illustrate the utility of Eikon by describing the advantages and disadvantages it offers the case study participants.

The objective of the case study is to critically examine the process of generating, and the nature of the output produced by Eikon with the goal of demonstrating that it allows the users to generate images that allow them to see aspects of the data that would have been difficult or impossible to see using other methods.

1.5 Thesis Structure

The remainder of this thesis is presented in eleven chapters:

- Chapter 2.

  Chapter 2 presents the fundamentals of image construction for multivariate data display. The properties of the visual variables of an image are presented and requirements for a multivariate data display system are discussed.

- Chapter 3.

  The various methods used by current visualization systems to present multivariate data are reviewed in chapter 3. Several methods of using color, geometry, and texture to display data are described, along with the limitations of each method when displaying multivariate data. I also present on-going research on how these limitations are being addressed.

- Chapter 4.

Chapter 4 reviews several methods of data abstraction and display that have been previously used to address the spatial allocation problem associated with displaying large amounts of information.[4] I discuss the advantages and disadvantages of each method. Drawing on the ideas of data abstraction and multi-resolution display I propose a new multi-scale tool, Eikon, to address some of the shortfalls in the application of these methods to geometric coding of large multivariate datasets.

- Chapter 5.

  This chapter introduces the first part of Eikon; the Eikon Toolkit (ETK). ETK is a stand-alone toolkit that provides a programmatic interface for developing visualizations and allows the visualization programmer to use the full power and flexibility of the C++ language in developing custom visualizations.

- Chapter 6.

  Here I present the second part of Eikon; the visualization programming environment - the interface for designing visualizations within Eikon. This environment allows visualizations to be constructed rapidly and provides an interface for exploring their parameter space.

- Chapter 7.

  The case study is introduced in Chapter 7 by describing the details of the application domain: geophysical fluid dynamics.

- Chapter 8.

  Chapter 8 describes how elementary views are constructed using Eikon by placing a glyph at each data point with a small subset of the data.

- Chapter 9.

  In Chapter 9 how Eikon generates global overviews through abstraction and important is discussed.

---

[4] These methods are also often called detail and context, or focus and context techniques

- Chapter 10.

  This chapter shows how reducing the detail at a global level provides more detail at a local level and supports intermediate level views which show important features at a large scale and more elementary details associated with them. I also describe how composite images can be generated by combining the output of two (or more) abstraction functions.

- Chapter 11.

  Chapter 11 concludes the case study by summarizing the advantages and disadvantages of Eikon as it was used to display large multivariate data.

- Chapter 12.

  This chapter concludes this thesis with a brief summary of the work presented and a description of the conclusions drawn. I also identify and describe several interesting avenues for future work.

Chapter 2

Fundamentals of Multivariate Data Display

> *"Visual representations of evidence should be governed by principles of reasoning about quantitative evidence. For information displays, design reasoning must correspond to scientific reasoning. Clear and precise seeing becomes as one with clear and precise thinking."*
>
> *Edward Tufte* - Visual Explanations - *1997*

2.1 Introduction

In this chapter we describe the problems of achieving graphical integrity and graphical legibility in multivariate data visualization. Tufte [tuft83] defines these two problems as: (1) how can an image be constructed that represents the properties of the data both truthfully and accurately (i.e. graphical integrity) and (2) how can the image facilitate reasoning about the data it represents (i.e. graphical legibility). The material presented in this chapter provides a context for discussing previous work in multivariate display (Chapter 3 & 4) and stems primarily from the observational work by Bertin [bert83] and Tufte [tuft83, tuft91, tuft97] and the more experimental work by Cleveland [clev94].

2.2 Image Construction

The fundamental problem in visual representation is how to map the data visually so that the resulting images have integrity and are legible. To achieve these aims, we must understand the various properties of the data we wish to display and the properties of the visual variables used to display them.

2.2.1 Properties of the Data

When depicting information the following must be considered: the number of data points contained in the dataset; the number of variables that define each data point; the number of individual categories that defined each variable; and the semantics or level of organization associated with these categories [robe90, bert83]

Number of Data Points, Variables, and Categories

A single dataset can contain millions of data points, each defined by several variables with numerous categories. However, a single image, supports only a limited number of visual variables: x and y position, size, shape, orientation, intensity, color, and texture [bert83]. The problem is how to usefully represent the large amount of information that a dataset contains through a combination of the limited number of visual variables offered by an image.

Level of Organization

To generate the most effective displays, the level of organization associated with the individual categories of each data variable must also be considered. The level of organization defines how the set of categories that make up the values of each data variable should be interpreted. Generally three levels are defined:

1. Qualitative The individual categories of a qualitative variable are nominal in nature - for example the English names of different countries. These categories represent concepts of differentiation, allowing statements such as "this is different from that, or this is similar, to that". There is no notion of order among the different categories and no measurable quantity is associated with each. There-

fore they do not permit universally agreed upon orderings of the categories to be made or allow ratio comparisons between them.

2. Ordered

   An ordered variable is a variable for which there is a universally agreed upon order of its categories - for example the four seasons of each year. Ordered variables allow statements such as "one category comes before or after another category" or "one category is higher or lower than another category." An ordered variable is also qualitative in nature; however, it does not allow statements regarding ratios between individual categories.

3. Quantitative

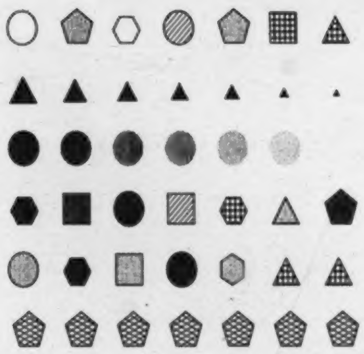
   Categories within a quantitative variable represent measurable magnitudes - for example, temperature in degrees centigrade. Such variables can support ratio comparisons. e.g, one category (or quantity) is twice another. Quantitative variables are both ordered and qualitative also.

2.2.2 Properties of the Image

Similar to data properties, the visual properties of an image can also be described in terms of the number of variables they contain, the number of possible categories per variable, and the level of organization supported by each variable.

Number of Visual Variables

In this thesis we focus on static two-dimensional images supporting a total of eight visual variables: x and y position, size, shape, orientation, intensity, color, and texture. Each of these visual variables has a number of distinct categories. Unlike data variables, however, the number of categories within each visual variable depends on a number of factors. These factors include: which visual variables are used in combination and the level of organization required by the data variables they represent. These factors also include the ability to produce and correspondingly the ability to perceive the visual stimulus itself. The visual variables x and y position, size, shape, orientation and texture all provide a large number of distinguishable categories which



In the second row, note how the decrease in size affects the viewer's ability to perceive the shape of the triangle. In the third row it is easy to order the circles from darkest to brightest. There are four squares scattered throughout this small image that can only be found by a serial search. The bottom row contains identical shapes rendered the same way at proximate locations. Despite the variation in position, it is possible to determine an association between them.

Figure 2.1: Properties of visual variables allowed by a static two-dimensional image.

are limited in part by the total area available for the image and the resolving power of the equipment used. Shape, size, orientation, and texture are further limited by the amount of space used to represent them. The range of intensity values is limited by the distance between the lightest and the darkest reproducible intensities as well as the overall size of the object. The number of categories supported by color is dependent upon the degree of saturation and intensity. All completely saturated colors are white and all zero intensity colors are black. In both instances, color is reduced to a single category.

Level of Organization

The level of organization of each visual variable is determined by its perceptual properties. Similar to data properties, three levels of organization can be defined.

1. Qualitative

   A qualitative visual variable allows us rapidly to identify instances of its occurrence in an image. Such a variable can further be described as either selective or associative:

   (a) Selective

A visual variable is selective if it is possible to identify and locate a particular instance of it without serial search of the image.

(b) Associative

Given a set of points uniform in all but one visual variable, a visual variable is associative if it is possible to ignore its visible variation across the set of points and reconstruct their uniformity.

Variation in position is selective. In Figure 2.1 it is easy to identify the shape that occurs in the top left position. Variation in shape however is not selective. It is difficult to determine the location of the four squares in Figure 2.1. Small variations in position are also associative. For example it is possible to group all the elements on the bottom row as associated even though they vary in position. Variation in size and intensity are, however, not associative. Variation in size or intensity affects the perceived visibility of other visual variables and therefore disrupts the uniformity of any information encoded by them. In the second row it is difficult to determine the shape of the object as being a triangle when its size diminishes. Also variation in intensity affects the perceived color, all zero intensity colors being black.

2. Ordered

   A visual variable is ordered if it is possible to reconstruct a universally common order of its values without recourse to a legend. Variation in position is ordered: objects at different locations can be ordered from left to right or from top to bottom. Figure 2.1 also demonstrates that variation in intensity and size are also ordinal. Variation in shape and color, however, are not ordered. There is no universally accepted order for shapes or colors and therefore the ordering must be provided by a data legend.

3. Quantitative

   A visual variable is quantitative if it is possible to determine the ratio of two data variable values it represents without recourse to a data legend. Variation in position and size are the only quantitative variables. Shape and color are not quantitative because they cannot be ordered. Intensity and texture variation are not quantitative as no standard unit of measurement exists. It's not possible to determine the ratio of white to grey or the ratio of a dense texture to a fine texture.

2.2.3 Data Display Models

The final step of multivariate display is to implement a method of correlating variation in the visual variable to variation in the data variables. This is accomplished by implementing a display model with a number of parameters and binding the data variables to these parameters. Variation in the data variables produces variation in the display parameters and correspondingly variation in one or more visual variables of the image. Display techniques in scientific visualization may be classified by the number of data variables a display model represents: scalar techniques represent data with one variable per data point [gonz93, kauf90, fren89]; vector techniques represent two or three variables (vectors) per data point [helm90a, rich90, ferr96]; and multivariate techniques represent n-variables per data point (discussed in Chapter 3). As the number of data variables increases, more creative ways must be sought to represent the data variables through the visual variables of the image. Visual variables must support the required number of categories for each data variable and the correct level of organization. If there is a mismatch between the number of categories needed by a data variable and the visual variable used to represent it, the data variable will be rendered less accurately than is needed. Similarly, a mismatch between the level of organization of the data variable and the associated visual variable can misrepresent the information the data variable contains. Both these factors affect the graphical integrity and legibility of the display.

2.3 Image Legibility

With an appropriate binding between data variables and visual variables, it is still possible to render an image that is not useful to the viewer. Just as a carefully composed written message can be rendered useless by illegible handwriting, an image can be ineffective if the relevant information is not visible or "legible". Beyond support for the required number of visual categories, several other factors influence the legibility of data in an image: the context in which the image exists; the ability to decode the binding between data variables and visual variables; and the types of information and, corresponding level of reading that are supported by the visual representation of the data.

2.3.1 Data Context and Encoding

A useful data image does not exist in a vacuum. Interpreting the rendered data necessitates that the viewer understands the larger context in which the displayed data exists, and appreciates the correspondences between data variables and the visual variables used to display them. The viewer must first select an appropriate display model and use an appropriate binding. A wide variety of both display models and possible bindings exist and the viewer must select a display model and binding such that truthful and legible images are constructed. This can be a difficult task and if done incorrectly can severely restrict the amount of information a viewer can extract from the visual representation [clev94] (Chapter 4) and can misrepresent the data in the image.

Although it may be important to isolate a small number of data variables for a particular experiment or simulation, it is equally important to be able to discover new information from the display and relate this information to the larger context (other data variables, other data displays, other experiments). To extract information from the rendered image, the viewer must know how each data variable has been represented; such information gives significance to structures or patterns that may be seen in the image.

2.3.2 Viewer Requirements

An image is useful if it meets the visual requirements of the viewer. No matter how wide and diverse the requirements are they share a common goal: to obtain insight into the displayed data. To obtain such insight, the viewer must not only understand what data is represented and how it is represented, but also be able to find answers to the queries they have about the data. The "most" useful image will answer all possible questions relating to the dataset. For large multivariate datasets, however, it is impossible to construct one single image that meets this ideal stipulation. An image constructed, for example, to reveal the relationships between one set of data variables may not be useful for other purposes. Therefore, from a legibility point of view, it is necessary to consider some notion of "important" or preferred queries and to generate images in which the answers to these queries are easily legible. This requires that the viewer have the ability to construct several different views of the data [clev94] (page 93.)

2.3.3 Levels of Reading

Some queries may be answered by examining a single element of the dataset in detail; others by examining small local groups of data points in less detail; and others through an overview of one or two specifics of all data points in the dataset [bert83, gall95]. The answers to these three types of queries are achieved through the ability to perceive the data at three corresponding levels of reading: elementary, intermediate, and global.

Elementary Level

At this most basic level of reading, attention is focused on a single or small set of data points, the data variables are displayed in detail, and it is possible for the viewer to recover their associated values with some precision. It has been suggested that text based tables of the data are more effective for elementary reading [desa84]. This may be the case for obtaining precise quantities for selected data variables, however, it may be more difficult to determine relationships between sets of variables, especially if many variables are associated with each data point.

Intermediate Level

The intermediate level of reading involves questions relating to groups of data points and data variables within the dataset. Generally the data are displayed with less precision than in the elementary level - this is done by reducing the number of data variables, reducing the number of categories for each data variable, or by displaying other characteristics derived from the original dataset. The goal of this level of reading is to see patterns and structures within local neighborhoods of the data, focus on a specific set of features, and compare them with patterns and structures in other neighborhoods. The ability to compare and contrast small sets of features from differing localities can help in understanding the global phenomenon represented by the data in the image.

Global Level

The global level of reading is similar to the intermediate level except that all data points are considered and typically little fine detail are required. This reduction allows global patterns and structures that are associated only with a few or even one particular feature of the data to be easily seen. Global levels of reading allow the entire dataset to be compared and contrasted with other global information from the same or different datasets.

2.3.4 Levels of Organization

The level of organization required by a particular data variable, and the level of organization supported by the associated visual variable determines the level of reading supported for the data variable in the image [bert83]. If a data variable is represented by a visual variable with the correct level of organization, elementary, intermediate, and global levels of reading may be possible. If the level of organization is wrong, only elementary levels of reading are facilitated, intermediate and global levels are more difficult to attain, and errors in judgment are more common [clev94]. For example, if an ordered variable is represented through shape variation, the viewer must continually refer to the data legend to reconstruct the order. This makes it difficult to determine accurately the ordering represented at intermediate and global levels where a variety of shapes may be visible. If the ordering is represented by intensity or size, then the intermediate and global order can be recovered without recourse to the data legend.

The most useful image enables all three levels of reading. However, the necessary reduction in information at each level limits the questions that can be answered. As a result the viewer needs to be able to construct multiple images that answer different questions at different levels of reading.

2.4 Conclusion

The most useful images for multivariate display are those that truthfully and accurately represent the data and facilitate reasoning about it; that is, images that exhibit both graphical integrity and graphical legibility. These images make correct and effective use of the limited visual variables to meet the viewer's large and diverse set of requirements, and allow the viewer to rationalize about the data at elementary, intermediate, and global levels.

Displays are accurate only when a data variable is represented by a visual variable that supports the correct number of categories and levels of organization. Of the eight visual variables, the x and y position variables are the most expressive: they have a large number of visibly distinct categories, and support all levels of organization. All other visual variables are adversely affected by size or intensity. As size decreases, the number of distinct categories for other visual variables decreases while the number for the positional variables increases. The reduction in size affects the perceptual properties of the non-positional variables and hence they do not support all levels of organization. In the next chapter I describe previous work with respect to how the various visual variables have been used to generate images of data.

For large and multivariate data, it is not possible to generate a single image that can provide an answer to all viewer queries at all levels of reading. It is necessary to generate several images with each image tailored to a small set of important questions. It is therefore important that any multivariate data display method easily allow the viewer to construct many different views of the data, and focus attention on various aspects of the data. Due to the large amount of information that needs to be displayed, the visual variables must be used efficiently. Due to the evident expressive power of the position variables, special attention should be paid to them. In this thesis I refer to these methods as spatial utilization methods to highlight the important role of the x and y positional variables. Elsewhere these have also been referred to as detail-in-context or focus-in-context methods. Previous work in this area is described in Chapter 4.

Chapter 3

Visual Coding

> *"If one does not see what one does not see then one does not see that one is blind."*
>
> *Paul Veyne*

3.1 Introduction

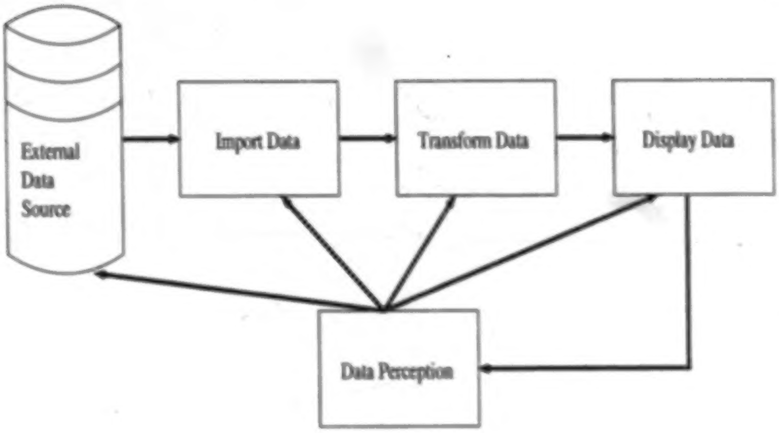
"Computers display and humans visualize."[1] This statement captures the essence of using computers to select and display data in ways that engage human visual processing to aid in their understanding. This subject involves research from many fields other than computing science, such as human performance and perception, logic, cognitive psychology, linguistics, human computer interaction, semiotics, and sociology. Computer science plays a central role as it provides the materials and methods for constructing visualization systems that are central to facilitating the visualization process.

Visualization Systems

Figure 3.1 illustrates the data flow pipeline, or visualization pipeline, on which the architecture of most visualization systems is based. As presented by Haber and McNabb [habe90], data acquired, for example through computational methods or by direct measurement, flows through a pipeline of connected processes. The data from an external data source is imported into the system and is typically subjected to

---

[1] Alan Fournier, 1992



Visualization can be characterized by an iterative process of transformation, display, and perception.

Figure 3.1: The visualization pipeline.

various transformation where particular data is selected or new data is computed. Attributes of the data are then displayed by mapping them onto various properties of a data display model (or abstract visualization object [habe90].) Finally the data is displayed by "executing" the display model to produce the image specified by the user. The process typically does not stop with the production of the image as the image may suggest new ways of looking at the data or new data to be examined.

Although the visualization pipeline presents a single abstraction of the structure behind different visualization systems, real visualization systems vary substantially on how this abstraction is implemented. One method by which visualization networks can be created is directly through programming languages, such as C++, using the graphics capabilities offered by APIs such as OpenGL [woo99] or Microsoft's DirectX [sanc00]. This type of approach allows programmers to use the complete functionality of the programming language to realize their visualizations. With this degree of flexibility and power however comes an increased difficulty in actually creating the visualizations, understanding the data flow and processes within the network, and exploring the range of parameters that maybe supported by the implemented visualization network.

This "low-level" programming approach lies at one end of a spectrum of approaches for constructing visualizations. At the other end lies specialized "turnkey" visualization systems that are specifically tuned to support the visualization of data from specific domains. Some examples include VolVis [avil94] designed for volume visualization; DataVisualizer [Wav90], presenting a menu driven application for displaying 3D data; the FAST system [banc90] and the FieldView system [lege91], for computational fluid dynamics; and the VIS-5D system [hibb90] for multi-dimensional earth science data. Such systems are very useful when applied in their respective domain, but are restrictive when data from other domains must be displayed.

More recently modular visualization environments have become very popular as an alternative to low-level programming or turnkey applications for constructing visualizations. Examples of such systems include aPE (from Tara Visual Inc) [dyer90], IRIS Explorer (from Silicon Graphics Inc.) [Sil91] AVS (from AVS Inc.), [upso89], Data Explorer (from IBM Corp.), [IBM91] Khoros (from Khoral Research Inc.), [kons94], and VTK (from Kitware Inc.) [schr98]. Within these systems visualizations are constructed (often with the aid of a visual programming interface) as networks of interconnected "modules." The networks represent data flow programs where data flows between modules over the interconnections and modules perform the various tasks at different stages within the visualization pipeline. These systems attempt to achieve the power and flexibility of low-level programming by allowing new modules to be added to the system whenever new functionality is required, but reduce the time it takes to develop the visualization by presenting a higher level interface. Often, however, the net result is that a large number of modules are available which, combined with the lack of detailed knowledge about the techniques they encompass, can make the construction of visualization networks difficult. In addition, the structures (data and control) imposed by the architecture of the visualization environment can negatively affect the implementation of functionality possible through low-level programming approaches.

The previous description categorizes visualization systems based on *how* the various stages of the visualization pipeline are supported or implemented. Although this provides some insight to the potential power and flexibility of each system it is more useful to categorize systems based on *what* they do to actually create the image of the data - i.e. the techniques they employ to filter, map, and display the data.[2] In the remainder of this chapter I describe numerous display models which have been proposed as ways of mapping and displaying data. In the next chapter I describe various methods for filtering and transforming data that have been developed to address the problems of displaying large quantities of data.

Display Models

The last ten years have seen a tremendous growth in the amount of research conducted into developing display models for visually encoding data. For the purpose of analysis and review, I describe this research in terms of three classes, depending on how the visual variables have been used to encode the data:

1. Color Coding (Section 3.2)

   The color of individual or small groups of pixels can be used to encode the data. Using this method data variation is represented as variation in hue, saturation, and brightness over the image.

2. Geometric Coding (Section 3.3)

   Parameterized geometric objects, or glyphs, can be used to represent individual data points. Data variables are bound to the various parameters of the glyph, causing data variation to be represented as variation in position, size, shape, orientation, intensity, and color of the glyph.

3. Texture Coding (Section 3.4)

   Texture coding represents data by variation in the perceptual properties of texture, such as size, frequency, orientation, density, regularity, etc., of features in the image. Texture can be synthesized directly from the data or generated using specially designed geometric glyphs.

Data Legibility

Once a coding scheme has been chosen, visual variables must be utilized so that the information displayed is both legible and accurate. Legibility of information in the image is determined by the viewer's ability to perceive different characteristics of individual data points (elementary reading), groups of data points (intermediate reading), and all data points (global reading). Data legibility is also affected by the number of data variables represented, the level of organization associated with each variable, and the total number of data points (refer to Chapter 2). In the next three sections I review and analyze previous work in data display with respect to the data and visual properties discusssed in Chapter 2. I conclude each section with a discussion of some interesting research problems.

---

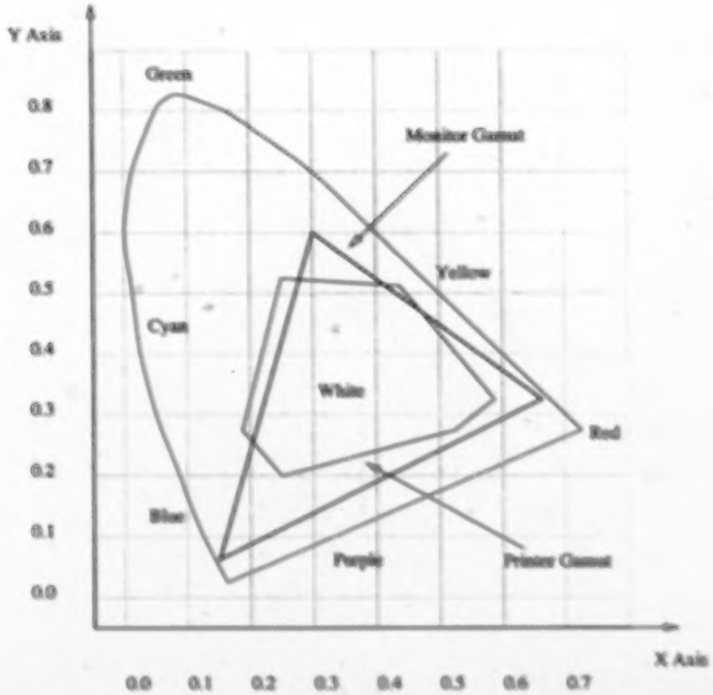
[2] Similar to Wilkinsons's [wilk00] description of specification, assembly, and display

3.2 Color Coding

Color is the perception of a distribution of electro-magnetic energy, and can be described by three parameters - the dominant wavelength, the excitation purity, and the luminance [fole90]. These three physical parameters of color correspond roughly to the notions of color hue (H), saturation (S), and intensity or value (V) respectively, and form the basis of the HSV color model proposed by Smith [smit78]. The HSV model, which is arguably easy for humans to understand, however, does not specify colors in a format that can be produced directly on color display devices. For colors to be produced on color cathode ray tubes (CRTs), for example, the HSV values must be converted to the RGB (red, green, blue) color model. The RGB model is based on the tri-stimulus theory of color perception [maca70] that describes colors as weighted combinations of the primaries red, green, and blue. Color CRTs produce color by additive combinations of the red, green, and blue light emitted by the CRT's electron gun. For colors to be produced as printed color images, the color specification must be converted to the CMY (cyan, magenta, yellow) model ([fole90] page 587) that describes colors as a weighted subtraction of cyan, magenta, and yellow.

Ranges of Colors

The color gamut of the RGB and CMY models are not identical, and neither can produce all visible colors; for example, using red, green, and blue primaries requires negative amounts of red to produce some colors, which is not possible on a color CRT. To analytically compare color models and their gamuts the Commision Internationale



Approximate gamuts for a color printing press and a color monitor are illustrated. Colors within the monitor gamut but outside the gamut for the printer cannot be reproduced on the printer. Similarly colors inside the printer gamut but outside the monitor gamut cannot be reproduced on the monitor.

Figure 3.2: The CIE chromaticity diagram and color gamut comparisons.

de l'Eclairage (CIE) developed the XYZ color model [l'ec32] that specifies all visible colors as weighted sums of three virtual primaries called X, Y, and Z.[3] By projecting the X, Y, and Z weightings of each color onto the plane $X + Y + Z = 0$, a two-dimensional chromaticity representation of the color space can be produced. This diagram is independent of luminance information: under this projection, colors with identical chromaticity values but different luminance values project onto the same point (see Wyzecki and Stiles [wysz82]). The chromaticity diagram provides a method of visualizing the gamuts of particular color models. Any three colors plotted on the chromaticity diagram can be mixed to produce any colors within the triangle they define. Figure 3.2 shows a comparison of approximate color gamuts for a color CRT and a color printer. This illustrates the potential problems involved in accurately reproducing color images viewed on the CRT on this particular color printer. Colors outside the gamut of the color printer will not be reproduced correctly.

Perceptual Uniformity

Perceptual uniformity is a desirable property for data display [taji83, meye80] as the Euclidean distance between two points in the color space is a measure of the perceived color difference between the colors they represent. The HSV, RGB, CMY, and XYZ models are not perceptually uniform. In the HSV model, for example, all colors displayed with intensity 0.5 are not perceived to be equally bright. The CIE has developed an approximately perceptually uniform space called the CIE $L^*u^*v^*$ color space [l'ec71][4]. This model describes colors using a perceived brightness factor L and chromaticity u^* and v^*, thereby maintaining the familiar concepts of hue, saturation, and brightness. These factors make the CIE $L^*u^*v^*$ model attractive for data display; however it is only approximately perceptually uniform, as it is based on results of color-matching experiments obtained under controlled conditions. The uniformity of the color space is not guaranteed under different conditions [robe86].

---

[3] These primaries are virtual because they are not actually visible as colors but can be mixed to produce visible colors.

[4] The $L^*u^*v^*$ model is recommended for modeling color viewed on CRT type displays. Another model called the CIE $L^*a^*b^*$ model has been defined for viewing color as reflected light. Details can be found in [wysz82] page 166.

3.2.1 Color Utilization for Data Display

The use of color look-up tables (LUTs), where entries in the LUT define the RGB values for individual colors, is the most common method of utilizing color for data display [schr98, luca92, upso89]. A mapping between color and data variables is defined by using the values of a selected data variable as an index into the LUT. The simplest way of constructing a LUT is to use slider bars to specify the values for the color component of each table entry. This technique is inefficient for specifying more than a few entries, and a more efficient method uses transfer functions to directly relate values of each color dimension to values of a selected data variable. Guitard and Ware [guit90] present a method that allows transfer functions to be interactively plotted for each color dimension by interpolation between a number of user defined control points or by drawing the function free-hand. Such techniques not only allow the mapping between a large number of data values and colors to be specified quickly but also allow better control over the properties of the color sequences defined.

Parametric Curves, Surfaces, and Volumes

The method described by Guitard and Ware (which uses the equivalent of one-dimensional parametric curves) can be generalized to bivariate and trivariate data by using two-dimensional parametric surfaces and three-dimensional parametric volumes. Parametric surfaces and volumes are attractive as they allow a mapping between data values and colors to be specified and manipulated using a relatively small number of control points. Pham [pham90] presented an example of this method using B-splines to model curves, surfaces, and volumes. He shows how B-splines can be used to generate a number of color sequences that have specific shapes and contain color sequences that have specific perceptual properties (see Section 3.2.2.)

3.2.2 Color Coding Legibility

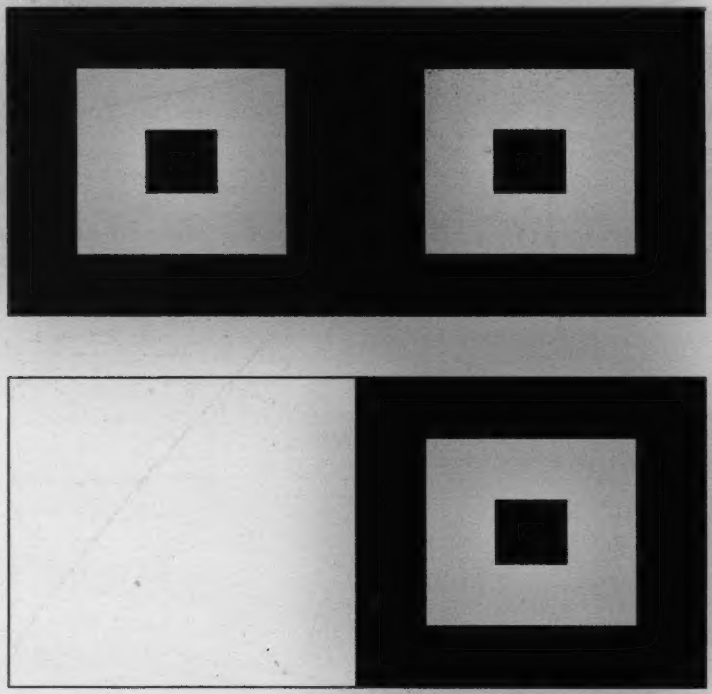
A number of researchers have investigated the ability to read color-coded images at elementary, intermediate and global levels. Wainer and Francolini [wain80] investigated the legibility of a bivariate color coding that used variation in saturation of red and blue to represent bivariate census data.[5] They showed this color coding to be poor for elementary, intermediate, and global readings of the images. Subjects made many errors in estimating the values at particular locations and made even more errors when the data legend was removed. To read the data encoding, subjects continually had to refer to the legend - this made intermediate and global readings more difficult as it required an "internalization" of the mapping describe by the legend. The phenomenon observed by Wainer and Francolini can be described by the need to use separable and integral visual variables as described by Garner [garn81]. To observe univarate characteristics of a multi-variate dataset it is necessary to encode each variable through visual stimuli that are separable. Also to observe multi-variate characteristics of many variables intergral stimuli must be used. The variables used to represent the bivariate census data - Hue and saturation - have been shown not to be easily separable.

Simultaneous Contrast

Legibility of color sequences has also been investigated by Ware [ware88a] whose experimental evidence suggests that elementary quantitative reading of univariate data can be adversely effected by the simultaneous contrast effect [boyn79]. The simultaneous contrast effect describes how two identical colors can be perceived as different depending on the colors that surround them. Figure 3.3 shows two examples of this effect. Ware's results show that more accurate quantitative elementary readings are obtained with color sequences containing few simultaneous contrast effects (such as a sequence approximating the color spectrum) than with sequences that induce more simultaneous contrast (such as a red to green sequence, or sequences based on luminance or saturation). Using the same color sequences Ware also showed that sequences based on luminance better depict global structural or form information than sequences based on the color spectrum; this he attributes to the fact that humans use variations in shading to determine shape. Ware also tested a combined sequence, based on the color spectrum and a variation in luminance to determine if both elementary quantitative and global form information could be read. The results showed that elementary reading is supported but did not conclusively show that global perception of form is also supported.

---

[5] There was special interest in this color coding at the time because it had been used by the U.S. Census Bureau to display bivariate maps.



In these examples, the gray ring looks brighter on the black surround than on the white and the pink ring seems redder with the red surround. These are examples of simultaneous contrast effects changing the perceived brightness and saturation of displayed colors.

Figure 3.3: Examples of simultaneous contrast effecting brightness and saturation.

Further Guidelines

Trumbo [trum81] has also presented work determining guidelines for effective color sequences that support various levels of reading. He presents four principles and sample color sequences based on these principles:

- Principle I - Order

  Principle I follows directly from the guidelines discussed in Chapter 2: a data variable with a required level of organization must be represented by a visual variable that supports the same level of organization. Principle I states that ordered information in the data must be represented using a perceptually ordered *color sequence*.

- Principle II - Separation

  Principle II also follows from the guidelines in Chapter 2: the required number of categories contained in a data variable must be supported by a visual variable with the same number of visually distinct categories. Principle II states that enough color variation must be used to support perception of individual or groups of data values.

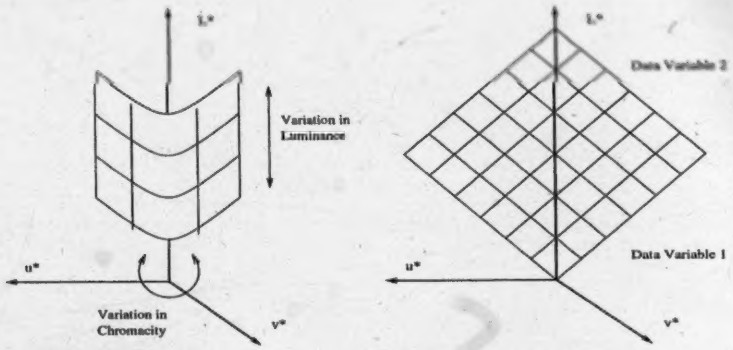
- Principle III - Rows and Columns

  Principle III and Principle IV deal with bivariate data. Considering such data as a rectangular data matrix where the rows and columns enumerate all the possible values for each variable, Principle III states that the rows and columns of this matrix must be distinguishable from one another, i.e., separable visual stimuli must be used that show the univariate characteristics of each variable.

- Principle IV - Diagonal

  Principle IV deals with identifying correlations between the two variables. For correlations to be evident it must be possible to distinguish colors that appear above, below and on the main diagonal of the data matrix.

Trumbo, Robertson and O'Callaghan [robe86] and Pham [pham90] present a number of color sequences that adhere to the above principles. Principle I is best achieved by using a straight line paralleling the luminance axis while; Principle II is achieved by using a straight line traversing several hues. Trumbo points out, however, that these two principles are diametrically opposed: as luminance is not effective for separation, and hue is not perceptually ordered, i.e, it appears difficult to have both separation and order in a single color sequence. This idea is also supported by Ware [ware88a] as described above. Principle III is achieved by using a segment from the surface of a cylinder whose axis is aligned with the luminance axis as shown in Figure 3.4(a) In this scheme, one variable is represented by a variation in chromaticity, and the second



(a) Principle I (b) Principle IV

Two color schemes that follow Trumbo's Principles III and IV, describing the need to maintain the univariate information of the Rows and Columns (Principle III) and distinguishing correlations between variables (Principle IV).

Figure 3.4: Examples of color schemes following Trumbo's Principles III and IV.

variable by a variation in brightness. Principle IV is followed by using a square at a fixed hue and aligning its diagonal along the gray axis (Figure 3.4(b)), thereby allowing distinction between data on, above, and below the diagonal. Again, neither of these schemes simultaneously supports Principle III, and Principle IV. Pham shows how parametric curves and surfaces can be used to specify these color schemes, and Robertson and O'Callaghan show how Trumbo's schemes can be implemented in CIE $L^*u^*v^*$ space. Using the surface of a double cone aligned with the lightness axis (as shown in Figure 3.5) they also describe a fourth scheme that follows Principle IV and shows both positive and negative correlations.

Dynamic Color Change

The previous methods all use fixed or static color sequences to represent features in the data. It is difficult for a single representation to convey all the necessary information and as a result research by Rheingans and Tebbs [rhei90, rhei92, rhei97] has investigated the possibility of using dynamically changing color sequences to gain further insight into the data. Using this method, users can visualize data represented by one color sequence, interactively change the color sequence, and in real time see the color coding change from one sequence to the other. Experiments were also conducted

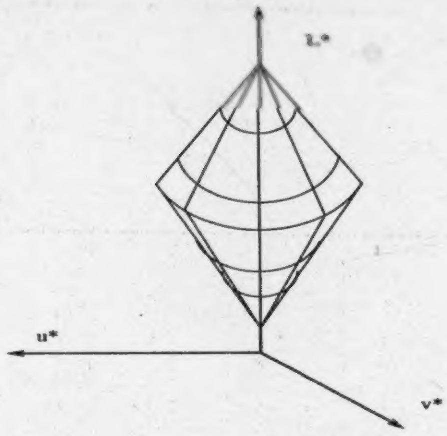


Figure 3.5: Robertson and O'Callaghans double cone scheme.

to determine the benefits of dynamic color change. In these experiments a bivariate dataset was coded in terms of two univariate color sequences and dynamic color change was supported by allowing the bivariate data to be represented as weighted compositions of both sequences. In two sets of trials, one using a fixed weighting of the color sequences, and one using an interactively variable weighting, subjects were required to determine the shape and magnitude of a number of features presented amidst varying degrees of noise. The results showed that users preferred the dynamic control and had higher degrees of confidence in their estimates. Using dynamic control as opposed to a fixed weighting users were more accurate in determining feature shape and at least as accurate at determining feature magnitude.

3.2.3 Future Work

In theory, color can be used to represent up to three variables in a single pixel, providing a great density of information. It is clear, however, that the perceptual properties of color make it inherently difficult to use. Several of these difficulties are surmounted by using uniform color spaces, adhering to guidelines for constructing color sequences, or by using dynamic color sequences. Other problems, however, still remain.

The color gamut for trivariate data display is limited. Bivariate color sequences are also difficult to design. Constructing univariate color sequences to represent quantitative and form information is easier to accomplish. For multivariate data display, color can be combined with other forms of representation to highlight some of the univariate characteristics of the data. Ware and Beatty [ware88b] present a system using spatially distributed squares differing in color to show coherence in a multivariate dataset. Work by Levkowitz [levk91] also shows how color, combined with geometry, can represent multivariate data through textured images.

Accurate reproduction of color images on multiple display media requires color sequences be completely contained within the gamuts of each display medium. Robertson [robe88] describes a system for modeling and visualizing different color gamuts. Multiple views of each gamut are provided, including three-dimensional volumes, two-dimensional slices, and one-dimensional plots of gamut characteristics.

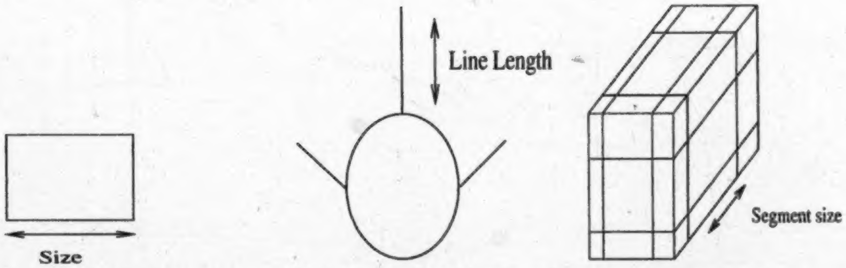
The large amount of research in color perception has produced a myriad of rules and guidelines that must be considered if color is to be used to its utmost affect. Recently a number of researchers ([rogo93a, rogo93b, berg95]) have developed a perceptual rule-based system to guide the user through color sequence design. Meta-data is used to describe important characteristics of the data and a set of perceptual rules determine affective color sequences.

3.3 Geometric Coding

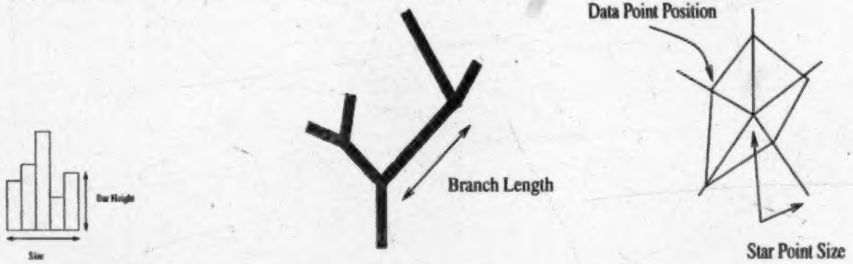
Geometric coding methods represent each data point as a glyph; this glyph is constructed as an organization of basic geometric primitives such as points, lines, and polygons. The various properties of these primitives are bound to data variables that in turn define each data point. Variation in the data is represented through alterations in the position, size, shape, orientation, and color of the geometric primitives that make up each glyph, providing a way to represent potentially large numbers of data variables in a single image. Several examples of glyphs are illustrated in Figure 3.6 and are described below.

3.3.1 Geometric Utilization for Data Display

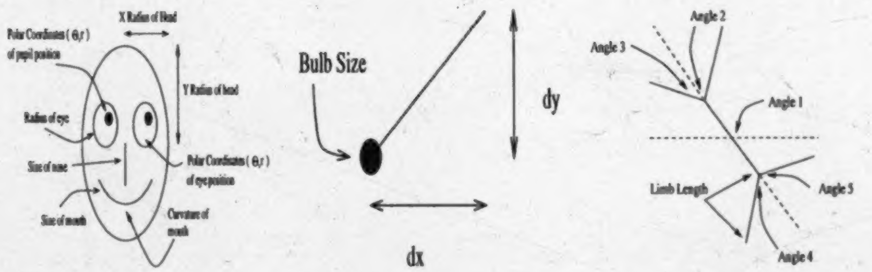
One of the oldest glyph techniques is the hedge-hogging technique for displaying vector data. In its simplest form, this method represents vector data using small



(a) Square Glyph (b) Circle Glyph (c) Box Glyph



(d) Profile Glyph (e) Tree Glyph (f) Star Glyph



(g) Face Glyph (h) Line Glyph (i) Stick Figure Glyph

Figure 3.6: Examples of several different glyphs as presented in the literature.

oriented line segments for each vector. It was first used by Edmond Halley in 1686 [hall86, tuft83] to depict the monsoons and trade winds of the globe. With one end point of the line used to show the data point location the remaining parameters of the line can be used to represent other data. The direction of flow can be depicted by orienting the line and tapering its thickness in the direction of the flow (as Halley did to show wind direction), the length of the line can be varied to show magnitude, and shadows can be added to depict the three-dimensional characteristics of the vector [klas91]

More complex forms of line geometry can be used to display further information. Wittenbrink *et al.* [witt96] used simple arrow glyphs where the width of the arrow head represented uncertainty in the data. Individual lines can be replaced with continuous flow lines integrated through the vector field. Polygons can be extruded over flow lines with variation in rotation, scale, and color used to represent flow characteristics [schr91]. Surfaces can be generated by triangulation of a set of adjacent flow lines [hult92] and volumes can be used to represent the three-dimensional structure of flow fields [max93]. Although primarily three-dimensional techniques, these methods give an indication how glyph-based techniques can be easily extended to display increasing numbers of data variables.

Circles, Boxes, Stars, Faces, and Trees

Glyph techniques have also been applied to more general multivariate data. In 1957, Anderson developed glyphs (called metro-glyphs) consisting of a small circle with a number of lines radiating from fixed points on its circumference [ande57]. He used variation in line length to represent data variation. To facilitate perception and recall, Anderson limited the number of lines to those emanating from the equator and the poles of the circle. To increase the legibility of the images, Anderson also limited the length of each line to values for short, medium, and long.

Rectangles have also been used to represent multivariate data. Ware [ware88b] uses a set of rectangles where the size and color of each rectangle represent variables from the data. This technique can be extended into the profile glyph [bert83] which, like a bar chart, uses of a set of rectangles to represent a number of variables. The idea of the profile glyph has been further extended by Kleiner and Hartigan [klei81] to display relationships between clusters of similar data variables. They represent each data point as a binary tree, where the leaves of the tree correspond to individual data variables. Each leaf is represented by a box whose size reflects the value of the associated variable. When leaves merge to form branches and branches merge to form the trunk, the average values of the data variables are used to determine the size. A similarity measure between data variables determines which leaves should merge to form branches and which branches should merge to form other branches. Kleiner and Hartigan present two glyphs based on this technique - they called these castles and trees. Castles are simply depictions of the binary tree representation of the data where the angles between the leaves and branches are zero, producing a glyph similar to the profile glyph. In the tree glyph, leaves and branches merge at given angles producing a traditional tree-shaped glyph, resulting not only in clusters of related variables being visible in individual glyphs but also in the ability to determine variation in different clusters across different data points.

Another glyph technique based on an orthographic projection of a three-dimensional box has been developed by Hartigan [hart75]. In its simplest form, the size of the sides of the box are used to represent three different data variables. To represent more data variables, each of the dimensions are divided into a number of different segments. By binding the size of each segment to the value of different data variables, individual segment size represents individual data variable magnitude, and the overall box size represents the overall data point magnitude. This concept has been extended by Alpern and Carter [alpe91] into a n-dimensional "hyper-box" for showing other relationships between data variables. Both box representations allow for simplification of the data by collapsing edges or by cutting along different axes. In this manner, a dataset can be represented by a collection of simpler boxes that can focus on particular features of the data.

Yet other geometric forms have been used to construct data glyphs. Siegel *et al.* [sieg71] have developed a star-like glyph that represents n data variables on an n-pointed star. For each data variable, an axis is drawn at a given angle from a common origin. The values of each data variable are then plotted on the corresponding axis and the points connected to form a star-like polygon. Values of data variables are read from the points on each axis and relationships between data variables and data points can be determined by variation in the shapes of the constructed stars.

Finally Chernoff [cher73] presents a technique that uses a two-dimensional representation of a human face to display each data point.[6] By varying head size and shape, nose length, eye position and orientation, and, mouth shape and length, his technique can represent up to eighteen different data variables. Chernoff suggests that by using facial representations, data clusters can be seen as collections of faces with similar properties. By using properties of the ears, hair, and facial lines, the basic "Chernoff face" has also be extended to represent even more data variables.

Customizing Glyphs

The previous methods give examples of only a few possible geometric configurations that can be used to represent data. Several recent systems allow users to design custom glyphs for data representation.

Ribarsky *et al.* [riba94] have developed a system called GlyphMaker that uses a simple three-dimensional graphics editor to organize graphic primitive such as points, lines, spheres, cuboids, cylinders, cones, and arrows. Selected properties of the glyph are displayed, and can be linked to different data variables using a "binding editor." The parameters of the binding are controlled by specifying minimum and maximum values for each data variable and glyph parameter. Data filtering is also supported by defining three-dimensional regions in space where data points are selectively displayed based on whether or not they meet the conditions associated with the region.

Another system developed by Mulder and van Wijk [muld95] also supports custom glyphs. In this system the binding between glyphs and data is controlled using a uniform interface of control points associated with each glyph. For example, the geometry of a cylinder can be specified using three control points: two for the central axis and one for the radius. Hierarchies of basic geometric objects can be defined by sharing control points. The system controls the behavior of components in the hierarchy by defining the degrees of freedom for shared control points. Associating a Cartesian coordinate frame with a control point using orthogonal x, y, and z coordinate axes generates one type of behavior; using a spherical coordinate frame using radius, azimuth, and elevation generates a different one. Mulder and van Wijk's system was also designed to support bi-directional binding between data and geometry. Bi-directional binding not only allows glyphs to act as output mechanisms by displaying data variables but also as input mechanisms to modify the data variables by direct manipulation of the glyph geometry.

---

[6] There is a certain irony in this as at the time A.I. practitioners were using numbers to distinguish between faces while Chernoff was using faces to distinguish between numbers

Beddow [bedd90] presents another system that supports automatic glyph representation of tabular multivariate data. The system computes small image matrices to represent each data point and tiles these over the display to generate the data image. For example given a dataset consisting of 625 data points, each defined by 25 variables, the data can be displayed by representing each data point as a 5×5 image matrix and tiling this over a 25×25 grid. The system calculates the average and standard deviation for each data variable and uses these to determine color ranges to show low, medium, and high values for each variable. This method produces images where similarities and differences in the data are displayed via variation in the shapes and colors of the data glyphs (5×5 matrices).

3.3.2 Geometric Coding Legibility

Glyph-based techniques provide easy methods of coding multiple variables in a single image. However, as with all visual coding, effective geometric representation depends upon ensuring that the required number of visibly distinct categories and level of organization for each data variable are supported. If a mismatch exists between the number of categories or level of organization required, the resulting geometric representation can degrade into a confusing and cryptic puzzle.

Intermediate and Global Reading

Chernoff [cher73] suggests that the humans ability to recognize faces allows us to select particular groups of faces based on different facial characteristics. However the complexity of the faces often makes it difficult to determine exactly what is being represented by variation in the facial properites and it has been shown [cher75, g.de86] that Chernoff may have been incorrect in his original premise that intermediate or global readings of the data are possible using differentiation in facial properites. It is also clear that it is difficult or impossible to discriminate between groups of faces if each individual face is small. Size has a dominant effect on the visibility of other visual variables and as size diminishes, the number of visible distinct categories of all other visual variables, with the exception of the x and y position, are decreased. Interestingly other work has suggested that the Chernoff face outperforms other geometric coding methods in particular circumstances [youn92, hait80]. Wikinson [wilk00] suggests that faces are better than other representations when the face is used to provide a "summary" of the data variables presented and are worse when each of the data variables encoded must be visible distinct. In this reasoning, similar to Garner [garn81], Wilkinson argues that the attributes of a face are non-separable and therefore perform well only when visual integration of the data variables is called for.

With careful design, some glyphs do facilitate the perception of intermediate and global patterns in the data. For instance, a line glyph tiled over the screen can generate a texture indicative of intermediate or global characteristics of the data The idea of using local geometry to induce global texture has been investigated by Pickett and White [pick66] when they represented student academic performance data with small triangle shaped glyphs. From the resulting displays, they noticed that individual geometric glyphs interacted with each other to produce a visual texture reflecting structures in the data beyond the extent of single glyphs. The patterns evident in the display illustrated the unusual fact that that honors students more resembled students who had dropped out than regular students.

This work was later extended by Pickett, Grinstein, and Smith *et al.* [pick88, grin89, smit91] who introduced "stick-figure" glyphs as an effective geometric code for generating texture. In their method data points are represented by multi-limbed stick figures, and data variables control the orientation and size of the limbs. Several configurations of stick figures are possible and when tiled over the display they generate different textures in which global characteristics of the data are visible. Unlike Pickett and White's triangle glyph, where elementary readings of the data are possible, elementary reading is more difficult with stick figures as they can interlock with their neighbors obscuring the visibly of some data variables. The legibility of images from both techniques also diminishes as the number of data points displayed increases. Each glyph must be drawn at a reduced size, which reduces the amount of detail made visible.

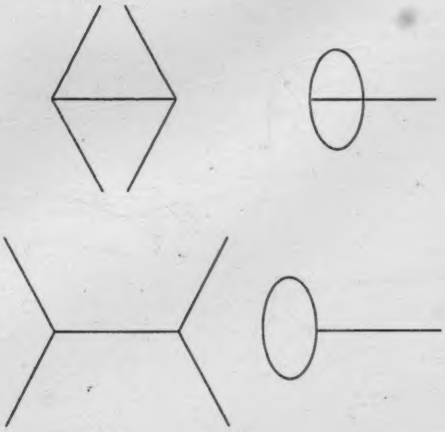
Levkowitz [levk91] combined color coding and geometric coding by representing each data point with a rectangular area of pixels. A number of line segments are used to divide the pixel area into a number of regions of different size, shape, and orientation. Colors can be assigned to the regions defined or to individual line segments. If colors are assigned to line segments, the color of each region is determined by interpolation between the colors assigned to the line segments defining the region. When these pixel areas are tiled over the image, a texture emerges where global structures in the data become visible. Levkowitz shows that by integrating several variables into a single display in this way, patterns become visible that are hidden when images of each variable are displayed alone. Again, however, the explicit use of geometry in this method renders it susceptible to problems of data density.

More recently, Laidlaw [laid98, laid99] has presented a system where color and multiple different glyphs are used to increase the amount of data that is legible in an image. In a technique, analogous to painting a picture, Laidlaw constructs visualizations by layering different images together. For example a suitable background color is washed over the image, on top of which another layer of glyphs (for example ellipses representing tensors) is composited. Other glyphs (for example line segments representing flow) can further be composited with the existing layers. As Laidlaw notes for this multi-layered multi-glyph display to be effective the glyphs and the data-glyph binding must be chosen carefully and the glyphs must be effectively distributed over the image. This notion is also re-iterated by another multi-layer technique presented by Crawfis and Allison [craw91] which is discussed later.

3.3.3 Future Work

Geometric coding is a useful mechanism for representing multiple variables in a single image. These techniques can be easily augmented with color [ware88b], can be used to produce texture [pick66], and can be combined with other non-visual stimuli such as sound and touch [ming95, volp97, hugh96]. Furthermore, geometric coding can be used not only as an output mechanism but also as an input mechanism suitable for computational steering.

Glyph representations can suffer from several "optical illusions" with effects analogous to the simultaneous contrast effect of color. Figure 3.7 shows some examples



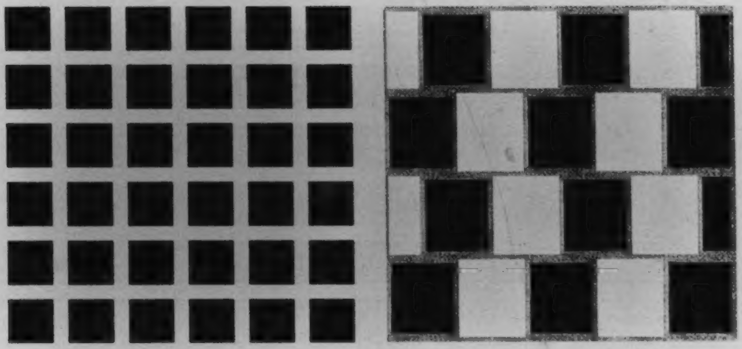
In all four images the horizontal line is the same length. In the two upper images it appears shorter than in the two lower images. This type of geometrical interference can cause errors in interpretation.

Figure 3.7: Local optical illusions due to geometry.

where elementary reading of glyphs may be affected and Figures 3.8(a) and 3.8(b) show examples where local geometry and shading introduce global optical illusions also.

As a result, the benefits of glyph-based representation are only fully realizable through the correct use of carefully designed geometric representations. Much research and development is still required into defining effective geometric codes for multivariate data. The line glyph, the triangle glyph, and the stick figure glyph are only a few examples of such codes. Research on texture perception, such as work by Julesz [jule81], Caelli [cael78], and Healey [heal98] may allow the definition of a number of principles that can govern the design of glyph representations. As with current research in color utilization, these rules could be used in conjunction with other automatic systems [shaw98] to remove some of the complexities in designing and using geometric codes.

The largest problem facing glyph representation is that the technique does not scale with dataset size. This problem can be understood by considering Wilkinson's [wilk00] comments indicating that abstract visualization objects can only be rendered perceivable through magnitudes in space. As more data is displayed, less space is available for each glyph reducing the visible variation that it can produce. At the



(a) Herman Grid (b) Cafe Wall [muns94]

Two examples of global optical illusions. In (a) spots of intensity can be seen between the squares, while in (b) the individual rows of bricks appear to taper towards each end. They are in fact level.

Figure 3.8: Global optical illusions due to geometry and intensity.

limit, in a raster based display, each glyph will be represented as a single pixel in which no geometric variation is possible. This problem is an example of the classic screen space allocation problem and several methods have been proposed to deal with it. These methods are discussed in Chapter 4.

3.4 Texture Coding

Rather than synthesize texture indirectly by using a geometric based glyph, texture can also be rendered directly from the data. Similar to color coding methods, these methods determine a color and intensity for each pixel in the image from a single (or group of) data points. The goal is to construct an image that contains a visual texture reflecting certain properties of the data. Directly rendering texture from the data in this way affords the advantage of not having to render thousands of small geometric objects and the ability to take advantage of current texture-mapping hardware to display images of the data rapidly.

3.4.1 The Dimensions of Texture

To use texture to display data, the appropriate dimensions of texture must be identified and systematic methods of producing texture based on data variation must be developed. Several experiments have been conducted to determine the salient dimensions of texture (Tamura *et al*[tamu78], Rao and Lohse [rao93]). Typically these experiments ask a number of subjects to classify a set of texture samples based on perceived features. The results of these experiments show that texture is a complex perceptual phenomenon governed by many factors, including perceived coarseness, repetition, contrast, shape, size, spatial distribution, linear-likeness, complexity, regularity, granularity, and orientation. The results also indicate that there is correlation between different sets of factors suggesting that all dimensions are not independent. Tamura *et al.* found high degrees of correlation between perceived coarseness, contrast, and roughness and also between orientation and line-likeness. Rao and Lohse also determined that texture may only have three principal dimensions corresponding to repetitiveness, a combination of directionality and contrast, and a combination of granularity, coarseness, and complexity. Haralick [hara73] also shows that texture perception is inherently related to scale. At one scale a coarse texture can be created using features that generate relatively low frequency changes in tone. At the same scale, a finer texture can be created by using features that generate higher frequency tonal changes. However as the frequency of change increases the predominant quality of the texture reverts back to tone.

By studying texture perception from a statistical viewpoint Julesz, Caelli, Gilbert, and Shep [jule81, jule73, cael78] have developed the texton theory of texture perception. They show that discrimination between textures having equal second-order statistics is not based on differences in higher-order statistics but on the presence of conspicuous local features called "textons." Experiments have identified several textons, including color, elongated blobs (linear features of specific width, orientation, and aspect ratio) and the number of terminators (line ends). Julesz also suggests that texture discrimination may only be based on the density of textons in the image. The texton theory is useful, as it demonstrates how useful visual textures may be generated based on local properties rather than on global statistics of the image.

3.4.2 Texture Utilization for Data Display

The most celebrated uses of texture have been Van Wijk's spot noise [wijk91, leeu95] and Cabral's line integral convolution (LIC) [cabr93] for displaying vector fields. Spot noise generates texture by using random intensity spots splatted at different locations over the plane and blending them with spots already drawn. Variations in the size, shape, scale, orientation, and intensity profile of the spots can be bound to data variables to produce directly texture features that reflect characteristics of the data variables. The spot shape can also be compressed, stretched, or bent to show features such as convergence, divergence, or curvature in the flow.

Like spot noise, line integral convolution can be used to produce texture images. This method represents the vector field by filtering a provided texture (normally band-limited noise) along flow lines from each point in the dataset. The intensity of pixels in the output texture are determined by the input pixel values convolved with a filter whose shape and length are determined by flow lines in the data. Several extensions have been developed for the basic LIC technique. Forssell [fors95] improves LIC on curvilinear grids by pre-warping the input texture to account for compression and expansion of the curved surface. Forssell also shows that phase shifted filter kernels are better than the periodic motion filters suggested by Cabral, for showing flow direction in an animated sequence of images. Stalling and Hege [stal95] present several optimizations for LIC based on using coherency between filter convolutions for pixels on the same flow line. They also show how multi-resolution LIC images can be achieved by varying the step size along each flow line. Kiu and Banks [kiu96] use multi-frequency noise to show flow magnitude where low frequency noise is used in areas of high velocity, generating long, fat streaks, and high frequency noise is used in areas of low velocity, generating short, thin streaks. Finally, Wegenkittl and Groller [wege97] show how flow direction can be displayed in a single image by using a kernel function ramped from low to high in the direction of the flow. They also present an inexpensive computational method that simulates LIC by using small disks of varying intensity and transparency placed along flow lines in the dataset. The use of disks in this manner is similar to glyph-based techniques, and, as a result, placing too many disks results in overlapping streaks obscuring the flow information.

Other texture synthesis methods that are suitable for data other than vector data have also been developed. Ware and Knight [ware95] describe a method of constructing visual texture using Gabor functions (the product of a sinusoidal grating and a Gaussian envelope). The orientation, size and contrast of features within the texture are controlled by binding data-variables to the position, orientation, scale, and amplitude of the Gabor function. Ware and Knight also describe what they call the uncertainty principle for texture. They show that due to perceptual qualities of texture, the size of texture elements for a given data density can be specified precisely but orientation cannot; conversely, orientation can be given precisely but not size. Therefore, as the density of data grows, it may not be possible to show all data variables precisely.

Finally, Crawfis and Allison [craw91] present a texture system that uses a 'C' like language called "Texl" to combine color coding, geometric coding, and texture coding to represent data. Color coding is supported by binding data variables to individual pixel colors, geometric coding is supported through functions for computing and drawing features in the data (such as two-dimensional contours), and texture coding is supported by using a "stamp" function that imprints copies of an input image or texture with varying orientation, scale, and transparency. The final output image is formed by compositing all three types of coding together. Crawfis and Alison noticed that compositing several images together in this way often generates images that obscure data features. To produce clearer images they allow individual colors to be modified based on data values, introducing a bump mapping effect into the image. The bump-mapping "lifts" the data out of the image, which the authors argue makes them less noticeable thereby increasing the visibility of other data. Clutter can also be created if too many texture stamps are made over the image. To reduce this type of clutter, stamping is done conditionally on the value of selected data variables.

3.4.3 Texture Legibility

The texture rendering systems of Crawfis and Allison suffer the same inability to scale with data size as do traditional geometric techniques. Direct texture rendering systems reduce the problem by implicitly filtering the data that they represent. As a result, they are not capable of showing precise detail about the data at elementary and often intermediate levels. A desirable property is the ability to "zoom" selected parts of the image and have more detail presented. This property is inherent in the use of glyphs but zooming texture generated images simply yields the same texture magnified.

Texture Dimensions

Although several studies have explored the different perceptual dimensions of texture, there is still some uncertainty about how many independent dimensions there are and in which way they can be used. Crawfis and Allison's system is capable of generating textured images that contain large numbers of variables; however, they have noted that such images are cluttered and that they obscure features of the data. Rao and Lohse, on the other hand, indicate that texture may only have three primary dimensions and several of the direct texture rendering systems display three or less data variables through texture. This suggests that texture itself may not be efficient for displaying larger numbers of data variables.

Multiple Views

Dynamic change has been recognized as important for data display. Color coding supports dynamic change through the ability to interpolate between selected color sequences for data display and geometric coding supports the design and utilization of multiple geometric forms to display the data. The limited number of direct texture synthesis models and the relative expense of computing different textures limit the use of texture for dynamically display the data. Gains may be made in rendering static images only at a cost of dynamic display.

3.4.4 Future Work

The field of texture coding remains largely unexplored and there are many avenues for future work. At the most fundamental level, we need to achieve a more complete understanding of the perceptual dimensions of texture and their utilization, and develop new synthesis techniques that will allow these dimensions to be used for data display.

Future research will involve experiments similar to those conducted by Rao and Lohse to determine independent perceptual dimensions of texture, and experiments building upon Wainer and Francolini's work with color to determine how such dimensions can best be utilized. Work by Healey and Enns [heal98] represents a recent example of such research; they examine the effectiveness of texture generated in terms of height, density and regularity of features in the image. The ultimate goal of this research is to produce perceptually uniform texture space models that describe in an orderly manner the possible dimensions of texture.

Another open research problem is the development of new texture synthesis methods that allow the various dimensions of texture to be bound to data variables. Computer graphics literature outlines many methods of generating texture and it would be interesting to apply some of these methods for data display. Of particular interest are procedural and multi-scale methods [perl85, eber94, perl95, heeg95]. Procedural texture methods could be used by binding data variables to the parameters of the procedure used to generate the texture. Multi-scale textures are interesting as they allow more detail to be added to or removed from the texture as the scale is changed. One problem with existing procedural texture methods is that often the link between the parameters and the textures generated is often not clear. To be useful for data display procedural, texture methods require a clear relationship between variation in the input parameters and variation in the texture. Any texture synthesis method must also be computationally efficient if it is to be applied to large datasets.

3.5 Summary and Conclusions

In this chapter I presented an overview of several techniques for visually coding multivariate data. This overview was based on three main methods of visually representing data: color coding, geometric coding, and texture coding. Within each method several techniques were presented and discussed in terms of the number of data variables they can represent, how the visual variables defined by each class can be utilized, and, how different factors affect the legibility of the images produced. For each technique a number of ongoing research problems where also described. The capabilities and limitations of each technique can be summarized as follows.

Color Coding

Color coding is effective at representing univariate data. Using color to represent bivariate and trivariate data is more difficult as the perceptual properties of color make it difficult to construct color sequences that can be read at elementary, intermediate, and global levels. Dynamically changing color sequences based on individual univariate color codings has been shown to be beneficial when displaying bivariate data. Overall color seems most useful when combined with other methods to represent multivariate data.

Geometric Coding

Geometric coding allows glyphs to be easily constructed as aggregates of simpler geometric primitives allowing the representation of data points that may be defined by large number of data variables. Glyphs can also be designed to facilitate intermediate and global reading of the data by generating a visual texture. Geometric coding methods do not scale well with dataset size. As the amount of data increases, more glyphs must be rendered, and the size of each glyph decreases, reducing the range of visual variation. This adversely affects the ability to read the data at all levels.

Texture Coding

Direct texture coding produces images of the data by synthesizing texture directly from the data without using an intermediate geometric form. This permits images of the data to be constructed without rendering thousands of geometric primitives and can produce overviews of large datasets compactly in a single image. Problems with texture coding include the lack of understanding of the dimensions of texture and their ability to represent information. The limited number of texture synthesis methods, coupled with the relative expense of computing different textures, make it difficult to dynamically explore data.

3.5.1 Conclusion

Geometric coding and texture coding can be regarded as lying at opposite ends of a visual spectrum. At one end, geometric coding provides the ability to generate detailed views of relatively small numbers of data points, while, at the other end, texture coding provides the ability to generate overviews of large numbers of data points. As the number of data points increases, the ability of geometric coding techniques to generate useful views of the data diminishes; similarly, as the number of data points decreases, the ability of texture coding to generate more detailed images containing more data variables is difficult.

There are a number of potential ways to solve both of these problems. The texture-coding problem can be addressed by developing multi-scale texture synthesis techniques, that dynamically introduces more features into the texture as scale permits. In this thesis I address the problem of scaling geometric coding methods to large multi-variate datasets. The scale problem is a well-known problem in data visualization and several potential solutions have been presented in the literature. These methods are discussed in the next chapter where various methods of transforming and filtering the data before it is displayed are presented and where their benefits and limitations are explored.

Chapter 4

Spatial Utilization for Data Display

> *"Everyone spoke of an information overload, but what there was in fact was a non-information overload."*
>
> *Richard Saul Wurma* - What-If, Could-Be - *1976*

4.1 Introduction

Displaying large amounts of information on relatively small display areas is a problem shared by many data visualization techniques. Naively displaying large volumes of information produces displays that are at best cluttered and unclear and at worst render no information visible at all. This problem is caused as a result of displaying too much information in too small a space and has been addressed in two ways: increasing the amount of space available for data display and reducing the amount of data displayed. In this chapter I present an overview of these methods (often referred to collectively as detail and context techniques) that I classify as scale-space methods and data abstraction methods.

1. Scale-Space Methods

   Scale-space methods manipulate the spatial dimensions used to represent data. These methods can be described as panning and zooming techniques, which interactively modify the number of data points and data variables displayed in a fixed space, or spatial deformation techniques, which locally adjust the amount of space available in order that data can be displayed in more or less detail. These methods are discussed in Section 4.2

2. Data Abstraction

   Data abstraction methods reduce the total amount of data that needs to be displayed. This is accomplished in ways ranging from displaying a single data point to computing multi-scale representations of interesting features.These methods are discussed in Section 4.3.

In this chapter I discuss the capabilities and limitations of these techniques in terms of glyph-based displays. I conclude by introducing a new system that follows from the previous work but presents a new method tailored towards multi-scale glyph-based representations of large multivariate datasets.

4.2 Scale-Space Methods

Scale-space methods focus on the utilization of the spatial dimensions to optimize data display. These techniques can be described by two broad categories: pan and zoom techniques and spatial deformation techniques. Pan and zoom techniques use panning to explore areas of the data and spatial zooming to explore interesting areas in more detail. Spatial deformation techniques use non-uniform allocation of space to render interesting data with increased detail and, correspondingly, less interesting data in less detail.

4.2.1 Pan and Zoom

Pan and zoom methods create multiple images in which the data is displayed at different scales. Panning and zooming methods are characterized by interactive techniques that allow the user to pan over the data and to zoom in or out on data of interest. When an area is zoomed, the amount of data displayed is reduced and is displayed on a larger scale, allowing more detail to be presented. Correspondingly when an object is un-zoomed or de-magnified a smaller scale is used and a less detailed view must be displayed.

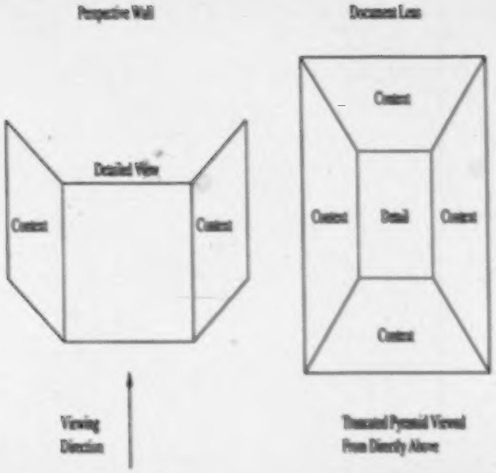
Star fields

Jog and Shneiderman [jog94] introduce a glyph-based system that uses panning and zooming to explore multivariate datasets. Two ordered data variables are used to determine the position of each glyph; panning and zooming being supported through two user interface components called "zoom bars" associated with the positional variables. Panning is supported by using the zoom bar like a traditional scroll bar and dragging it over the range of values for the positional variable. Zooming is supported by narrowing or expanding the range of data values represented by the spatial dimensions of the image. This range is computed relative to the zoom bar location in the associated data variable. Narrowing the range simulates zooming in, while expanding the range simulates zooming out. Dynamically updating the position of glyphs in response to zooming creates an an effect similar to flying through a "star field." At any particular zoom level the number of actual data points displayed can be restricted to those that fall within specific ranges of selected data variables. This allows the reduction necessary to display data on a reduced scale, i.e. when the user zooms out.

Pad and Pad++

Perlin and Fox's [perl93] Pad system and its successor Pad++ (developed by Bederson and Holland [bede94]) represent another panning and zooming system. These systems center around the use of portals to view objects or data. Portals describe how objects viewed through them should be transformed, magnified, and displayed. By nesting portals within portals the amount of magnification and transformation can be controlled. Different images of the data are generated by associating different graphical representations for each object with different zoom levels. Distinct representations can be used for non-overlapping zoom levels or procedural objects can be used that adjust the rendering based on the current zoom level.

Maintaining interactivity is important in all panning and zooming methods as it helps the user stay oriented in the data space. Pad++ supports interactive panning and zooming by employing hierarchical bounding boxes for visibility calculation, clipping based on portal extents, and rendering reduced detail during navigation.



The perspective wall is shown on the left and the document lens is shown on the right. Both methods integrate detail and context through mapping data onto a central screen (detail) and onto surrounding screens angled away from the viewer (context). A two-dimensional view of the data is achieved via a perspective projection of this mapping.

Figure 4.1: The perspective wall and document lens.

4.2.2 Spatial Deformation

Panning and zooming allow data to be displayed by rendering the data using a uniform scale in different images; spatial deformation techniques, on the other hand, allow the data to be displayed using a non uniform scale in a single image. This is accomplished by non-uniform allocation of space to different data regions and allows data in different data regions to be displayed using different levels of detail.

Perspective Wall and Document Lens

Mackinaly *et al.*'s [mack91] perspective wall represents a spatial deformation method designed for linear data spaces, that is, a data space that is much wider than it is high. A similar technique called the document lens has been proposed by Robertson and Mackinlay [robe93] for display and navigation of large document spaces. Both techniques use a central screen, perpendicular to the viewing direction, surrounded by a number of side screens angled away from the viewer to display the data as shown in Figure 4.1. Different areas of the data are associated with each screen, and by using a perspective projection an image is produced that provides both detail and contextual views of the data in a single image. The projection results in most space being allocated to the center screen (and therefore the data associated with it) as it is closest to the viewer, and decreasing amounts of space to the side screens (based on position along the screen and the angle to the viewing direction). In both techniques the user can pan over the data so that interesting data is displayed in detail on the center screen. The allocation and deformation of space can also be controlled by changing the size and position of the center screen and the angle of the side screens.

Graphical Fish Eye

Both the document lens and the perspective wall have discontinuities between the detail and context views. Several spatial deformation methods provide smoother integration between these views, such as the graphical fish eye view by Sarkar and Brown [sark92]. Their technique is based on the metaphor of a fish eye lens that distorts the image by magnifying objects at the focal point and uniformly decreasing the spatial extent of data with respect to its distance from the focal point. Originally developed for displaying large connected graphs, fish eye views use a global significance associated with each vertex and the distance of vertices from the current focal point to determine the size and position in the distorted view. Vertices close to the focal point with a high global significance are alloted more space than vertices further away with lower significance. Sarkar and Brown also use a function to compute the "visual worth" of a vertex that is used to determine if a vertex should be drawn or not.

Rubber Sheets and Pliable Surfaces

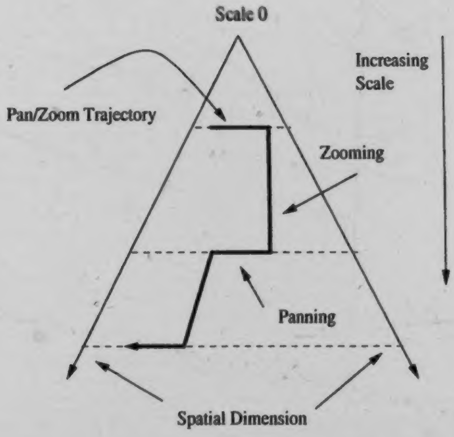
The graphical fish eye view supports distortion based on a single focal point. Other techniques allow the spatial dimensions to be distorted based on multiple focal points and focal areas. Sarkar *et al.* [sark93] present a method based on stretching a virtual rubber sheet that allows multiple polygonal focal areas to be defined and enlarged. This method and the graphical fish eye view can both introduce distortion into the image, which changes the relative position of data points displayed, i.e., a data point to the right of another can actually be displayed to its left. This problem is addressed by stretching orthogonal areas uniformly enlarging data within the focal area and uniformly reducing data outside, thereby maintaining the relative order between data points in the image. This however re-introduces large discontinuities between detail and context similar to the perspective wall and document lens.

Carpendale *et al.* [carp95] present a technique similar to the rubber sheet method but this time using the concept of a pliable surface. Detailed views are generated by pulling points on the surface closer to the viewer and rendering the surface using a single point two-dimensional perspective projection. This ensures that the areas pulled towards the viewer occupy more space in the rendered image. The surface generated by pulling points is modeled as a three-dimensional Gaussian curve that ensures smooth integration of detail and context. The height of the pulled point and the width of the curve both affect the distortion introduced into the image. Uniformly scaled areas can be created using a curve with a flat top and multiple focal points can be defined by blending multiple curves together. Like the rubber sheet method, the pliable surface, can adversely affect the relative ordering of data points displayed; however, unlike the rubber sheet, the pliable surface has an inverse mapping that allows the undistorted view to be recovered. The amount of distortion in the image can also be visualized by shading the surface used.

4.2.3 Scale-Space Diagrams

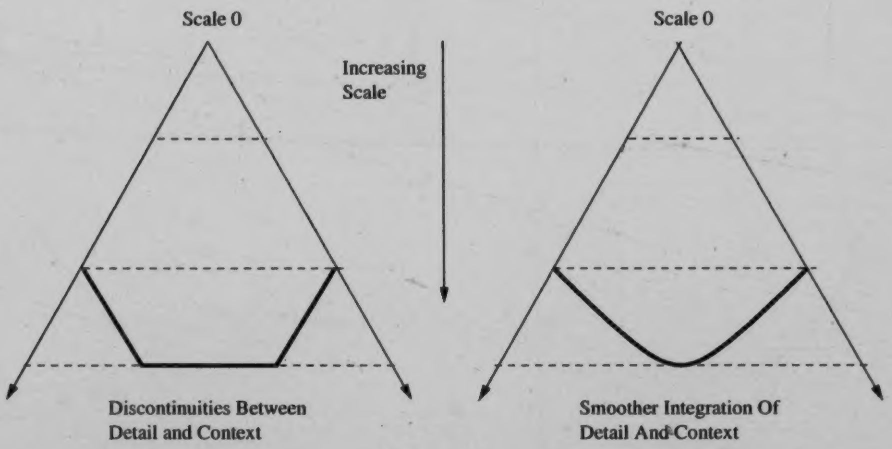
Both panning and zooming and spatial deformation techniques provide ways of generating detailed and contextual views of information. Panning and zooming provide detail and context by interactively generating a sequence of images showing different areas of the data on time varying scales. Spatial deformation methods provide integrated views where both detail and context are provided in a single image on spatially varying scales.

The characteristics of both techniques can be described using a scale-space diagram as presented by Furnas [furn95]. Scale and space are represented by an infinite pyramid where the horizontal dimension represents space and the vertical dimension represents scale. The apex of the pyramid corresponds to a scale of zero and each subsequent horizontal layer represents increasing scale. Panning and zooming can be described by particular trajectories through this space as shown in Figure 4.2. Pan-



Scale space is represented as an infinite pyramid with a scale of zero at the apex and increasing scale as we move down the pyramid. In this representation, panning is modeled as a trajectory through different spatial locations at the same scale while zooming is modeled as a trajectory through the same spatial location at different scales.

Figure 4.2: Panning and zooming in scale-space.


Scale 0
Increasing Scale
Scale 0
Discontinuities Between Detail and Context
Smoother Integration Of Detail And Context

Spatial deformation in a single image can be modeled as different areas of the data displayed on different scales. The image on the left shows a spatial deformation that has discontinuity between the views displayed at different scales while the image on the right depicts an image with smoother integration between the various scales.

Figure 4.3: Spatial deformation in scale-space.

ning is described by horizontal lines at a fixed scale, while zooming is characterized by vertical lines through several scales. Spatial deformation methods are characterized by lines and curves through scale-space, illustrating that data are displayed on several scales with varying degrees of continuity between them as shown in Figure 4.3

Scale-space methods are only a partial solution to the spatial allocation problem. When displaying large multivariate datasets using glyph-based representations, detailed views can be generated easily by rendering a glyph at each data point. Contextual views (or global views), however, are not possible without some form of information reduction or data abstraction. Each of the methods discussed here have some form of data reduction inherent in their description. In the remainder of this chapter I discuss various other techniques proposed in the literature for performing this reduction.

4.3 Data Abstraction

The importance of abstraction in terms of data display is not a new concept. In 1786, William Playfair [play86] remarked:

> "Men of great rank, or active business, can only pay attention to general outlines; nor is attention to particulars of use, any further than as they give a general information: And it is hoped, that with the assistance of these Charts, such information will be got, without the fatigue and trouble of studying the particulars of which it is composed."

The word abstraction is derived from the Latin *abstrahere* meaning to draw from. Data abstraction involves drawing away from the immediate details of the data to represent it in some generalized way, often generating a smaller dataset that captures some essential of the original. Graphical depictions of these abstractions can focus the viewer's perception on important aspects of the data, and remove the distraction of unnecessary detail [phil88, stro98]. Data abstraction can be looked at as the process of analyzing and searching data - data mining - with the objective of finding implicit but potentially useful information [keim94b, keim94a, keim96]. In this section I present an overview of data abstraction techniques that are applicable to glyph-based data displays.

4.3.1 Data Probing

The simplest form of data abstraction is to select and display a limited number of data elements from the original dataset. This can be done by directly selecting data points from the dataset, or applying a selection criteria to determine which data points to display. Data probing presents the most basic form of selective data display, displaying a local subset of the data around a focal point in the dataset. Glyph techniques can be readily applied as data probes with the benefit that the reduced number of data points allows the use of complex geometric forms to represent a number of data characteristics. De Leeuw [leeu93] demonstrates this by using a local probe to display several characteristics of a vector field around a user-selected focal point. Van Wijk and van Liere [wijk93] presented another technique that uses an $N \times N$ matrix of images to show relationships between all pairs of data variables from an N-dimensional dataset in the vicinity of of a user selected focal point. Both methods explore the dataset by interactively moving the probe through it, thereby achieving some degree of intermediate and global comprehension.

4.3.2 Autonomous Probes

Associating behaviors with data probes allows them to be used as autonomous agents to explore and display the data: the probe moves through the data, outputting new data or graphical primitives in its wake. This technique has been widely used in multivariate visualization. Van Wijk [wijk92], Stolk and van Wijk [stol91] and Bryson and Levit [brys91] present several examples of using particle systems [reev83] to display vector fields. Particles are introduced into the flow with behaviors that enable them to follow flow lines from the points of entry. The specific nature of the image produced is determined by the particle source position, particle source geometry, individual particle geometry, the rate of particle release, and the "life span" of each particle. Each of these attributes can be adjusted by the user to generate different representations of the data.

Pang *et al.* [pang93, pang97] have proposed another system based on autonomous probes. This introduces the idea of spray rendering, that simulates the "spraying" of smart particles or "sparts" into the data. Sparts search the data for interesting features and execute a specified behavior such as generating new data or a graphical representation when such features are found. Pang also developed the "Mix and Match" system [pang94] that provides a user interface for the specification of different behaviors for different sparts. Kerlick [kerl90] has also explored the the use of multiple probes for data visualization and suggests that enabling individual probes to communicate could further enhance the displays.

4.3.3 Conditional Display

Data probing is good at generating local or intermediate views of the dataset; however, a drawback of these methods is that it can be difficult to obtain accurate global pictures. To obtain a global overview the viewer has mentally to integrate a number of separate images and the probes must be placed in the correct locations so that the right information is displayed.

Furnas [furn86] presents a technique that combines the idea of data probing with conditional display of other data points to achieve a more global view of the data. This method not only displays data around a selected focal point but also displays other interesting data that may be more distant from it. In this scheme, a degree of interest (DOI) is determined for each point in the dataset as the difference between a global significance assigned to each point and a measure of its distance from the current focal point. Displays of the data are generated by only displaying data points with a DOI above a user-specified threshold. This generates views that provide detail in the vicinity of the selected focal point, and context by displaying more distant but globally significant data points. This produces what Furnas calls graphical fish eye views, and is similar to Sarkar and Brown's graphical fish eye method. (Sarkar and Brown's method is different in that they apply the technique to adjust the spatial extent of data displayed.)

4.3.4 Feature Extraction

Conditional display techniques can generate overviews of the data that contain more global information than could be easily obtained by using simple data probing. A disadvantage of these methods is that satisfying the condition to display a data point is an all-or-nothing decision. As a result, subtle but important variations in the data could be obscured: lots of data points that match the conditions would generate a cluttered view; displaying too few data points would make inefficient use of the display. *Feature extraction techniques attempt* to address this problem. Unlike previous methods, that display all variables or characteristics of a single or small group of data points, feature extraction methods attempt to display a few characteristics of all data points. These methods reduce the amount of data that must be displayed by determining the presence of interesting features and displaying only these in the image.

Feature extraction methods have found most application in the display of vector fields. Post *et al.* [post95] use this technique by computing interesting features of the flow, determining characteristic values for these features and using these characteristics to render the data. A similar technique uses the location and characteristics of critical points extracted from the vector field [helm90a, helm90b, hess97, glob91, helm91]. In these methods, critical points are distinguished as repelling or attracting foci, attracting or repelling nodes, saddle points, etc. [helm90b]. Critical points are displayed by associating a glyph with each critical point, and the characteristics of critical points can also be used to connect glyphs together to depict global flow topology.

The idea of feature extraction has also been applied to multivariate data in general. Assa *et al.* [assa97] construct images of multivariate data by representing each data point's similarity to a set of user defined features. Each feature is represented as a point in space and data points gravitate towards the features that they are most relevant too. This results in a low dimensional overview corresponding to a relevance map where distance between points and features indicate the degree of relevance of the data point to the displayed feature. One problem with such relevance maps is that when a cluster of data points are close to a set of features, ambiguity exists as to what data points are relevant to which features. Assa addresses this problem by using a set of features points that contain combined features. If it is found that a large number of data points are relevant to two common features then these features are combined into a new feature point. This helps to identify those points that are close to the new combined feature.

4.4 Conclusion

The design of display methods that can represent a variety of characteristics on a number of different scales from a wide variety of datasets has been identified as a worthwhile but difficult pursuit in scientific visualization [gall95]. With respect to large multivariate datasets, the ability to render different images of the data with the combined use of scale-space manipulation and data abstraction provides the key to this problem.

In this chapter, several scale-space and data abstraction methods have been reviewed. Scale-space techniques are useful interactive techniques for exploring data, however, to be applied to large multivariate datasets they must be used in conjunction with some form of data abstraction. Several examples of data abstractions applicable to glyph-based displays of multivariate data have been reviewed. These abstraction methods suffer from two main limitations that restrict their use for visualizing large multivariate data:

1. Limited support for global or intermediate readings.

   Data probing uses a single or small group of probes to display elementary information. Selective display methods are critically dependent on the conditions and thresholds used to generate the display. Graphical fish eye views and other focal point techniques are equally dependent on the location of the focal points. As a result, none of these techniques are useful for obtaining reliable global overviews of a dataset.

2. Limited support for elementary or intermediate readings

   Feature extraction methods are capable of generating global overviews of the data. However, they are less adequate for computing the detailed views required for intermediate and elementary reading. By definition these methods reduce the data to a small set of features and their characteristic values. More detailed views of this data will render more detail about the features and not necessarily about other information in the dataset. For example, Helman's technique [helm91] computes the location and characteristics of critical points. Using this method only more information about the critical points could be displayed and not information about other flow characteristics. Information regarding other features in the data could be generated by recursively applying different abstraction functions. This has been suggested by Post *et al.* [post95]. This method requires several different graphical representations for each abstraction, as each abstraction compute different features and characteristic values. Assa's [assa97] low dimensional overviews are amenable to a single graphical representation and more or less detail can be generated using more or less feature points. This method however utilizes the spatial dimensions to encode relevance and therefore destroys any spatially significant information in the data.

4.4.1 The Eikon System

In this thesis I present a new multivariate data display system, called Eikon, that stems from the principles of data abstraction (or data-mining) and multi-scale display. The purpose of this system is to use these principles to facilitate the use of existing geometric coding methods for visualizing large multivariate datasets. In this respect the primary goal of Eikon is not to replace or demonstrate that it is better than existing visualization systems (such as VTK, AVS etc) but instead to demonstrate methods by which the geometric coding techniques supported by these systems can be extended to large multivariate datasets.

Dataset Structure

Specifically Eikon has been designed to work on datasets where data points are defined via a structured points dataset (see [schr98] page 118). In such a dataset the data points are arranged on a regular, possibly multi-dimensional lattice with uniform spacing between each point. Such a regular structure is one of the more simple data structures in scientific visualization and it is also one of the most common. Also although the lattice can be multi-dimensional two dimensional and three dimensional[1] lattices are the most common. In these cases the two and three dimensions of the lattice typically represent spatial location and there is a certain amount of significance attached to the location of data points with respect to each other and within the spatial extent in which they are defined. Such datasets are common in areas such as computational fluid dynamics, medical imaging, earth and atmospheric sciences, and the like where data are generated or collected on a regular lattice representing some area of real world space. Currently Eikon is implemented to process instances of such two-dimensional lattices. Three dimensional and higher can be visualized by extracting two-dimensional slices and displaying these.

---

[1] Time varying lattices are also commonly encountered

Eikon is not suitable for processing data defined via unstructured points or grids, or regular grids that are not rectangle based. Unstructured points correspond to data measured at various different points in space with no obvious structure between them and no topology defined by them. Unstructured grids are similar except there is a defined but completely irregular topology that "connects" the data points together. These types of datasets are unsuitable because Eikon relies on a regular topology (a regular rectangular lattice) to define the structure of the dataset. Unstructured points and grids can be processed if they can be transformed into a structured points dataset (via sampling and interpolation for example.) In some cases such transformations are difficult, for example in the case of network data where the position of nodes (data points) and the links between them are meaningful.

Another common form of dataset structure are rectilinear and curvilinear grids. In these datasets the topology of the lattice is regular but the actual position of data points (the geometry of the grid) is irregular - i.e. the spacing between points may be irregular and the grid may be curved. Eikon can be applied to structured grids however a certain amount of error will be evident from the irregular geometry of the grid. This error will be small if the geometry approximates a structured point set and increases as the difference increases. More accurate results are obtained if the structured grid is first transformed to a structured point dataset.

The Process

Eikon is based on the use of multiple abstractions and the concept of multi-scale importance to define multi-scale images of the data. It utilizes existing (and can utilize future) glyph display models to generate multi-scale views of the data from broad global overviews to precise detailed elementary views. Eikon differs from other systems in that rather than displaying views from a computed lower resolution version of the data, it displays different views directly from the raw data present. Different views that facilitate global, intermediate, and elementary readings of the data can be rendered without the need for different display models. The legibility of global and intermediate views is also not critically dependent on the geometric properties of individual glyphs (although it can be enhanced if appropriate glyphs are used.)

Views of the data are generated based on the original data (or optionally some abstraction of it) by distributing glyphs over an image using a multi-scale representation of some user defined important characteristic. Elementary views can be displayed using individual glyphs; intermediate views can be displayed as groups of glyphs; and global views can be displayed as a spatial distribution of glyphs over the image. The glyph distribution process makes the display less sensitive to thresholding artifacts of other selective display techniques, generates views of the data that maintain the spatial relationships between data points, and allows various levels of detail to be produced by using a variable number of glyphs to display the data (and also reducing the rendering load placed on the hardware.)

The Interface

Eikon is designed as a modular visualization environment that couples the rapid prototyping possible with a visual programming environment (similar to AVS [upso89]) with the flexibility of a toolkit (similar to VTK [schr98]), and the power of low level programming. In this respect Eikon is a tool used by a visualization "expert" in conjunction with a researcher or scientist (the user) that allows visualizations to be created quickly, allows the user to see and manipulate the parameters of the visualization and, with the aid of the visualization expert, to add new modules to the system as new methods of displaying the data are determined - i.e. new methods of abstraction, importance specification, or new glyphs. The goal of the system being to allow the user to represent as many data items as possible on the display, allow the user to interact with this data (exploration), to allow the user to construct queries, or hypotheses about the data, and to guide these visualization activities based on visual feedback from the system [keim94b].

The architecture of Eikon is similar to that of VTK and it should be straightforward to incorporate the methods demonstrated within Eikon along with the visual programming interface into VTK in the future. The complete details of Eikon are presented in the next two chapters.

Chapter 5

ETK: The Eikon Toolkit

> *"The goal of effective representational image making, whether in oil or in numbers, is to select and manipulate visual information in order to direct the viewers attention and determine their perception."*
>
> *D. Phillips* - SIGGRAPH - *1988* [phil88]

5.1 Introduction

In the previous chapter I discussed several methods that address the problems of displaying large amounts of information. In general these methods adjust (1) the amount of data that must be displayed, (2) the scale at which it is displayed, or (3) the amount of space allocated to regions of the dataset. In this chapter I describe Eikon, a system that combines several of the concepts of data abstraction with methods of using scale and space to represent multivariate data. My system is based on the observation that previous glyph-based techniques support data variation through altering size, shape, and orientation, but typically don't encode data by changing the spatial density of glyphs over the image. In terms of visual coding, the spatial dimensions are regarded as the most expressive. My system takes advantage of this and renders large datasets by selectively placing varying numbers of glyphs at different locations depending on the characteristics of the data found there. My system is also based on the premise that attempting to display all the information contained in the data may be the wrong approach. Displaying all the data requires a tremendous amount of rendering power and results in little emphasis being placed on any particular attribute of it. Eikon is based on detecting selected features of interest at multiple scales within the data and using this multi-scale structure to distribute variable number of glyphs over the image. Using this method global overviews of large datasets can be generated using variation in glyph density. Geometric variation of individual glyphs can also provide more detailed intermediate and elementary views.

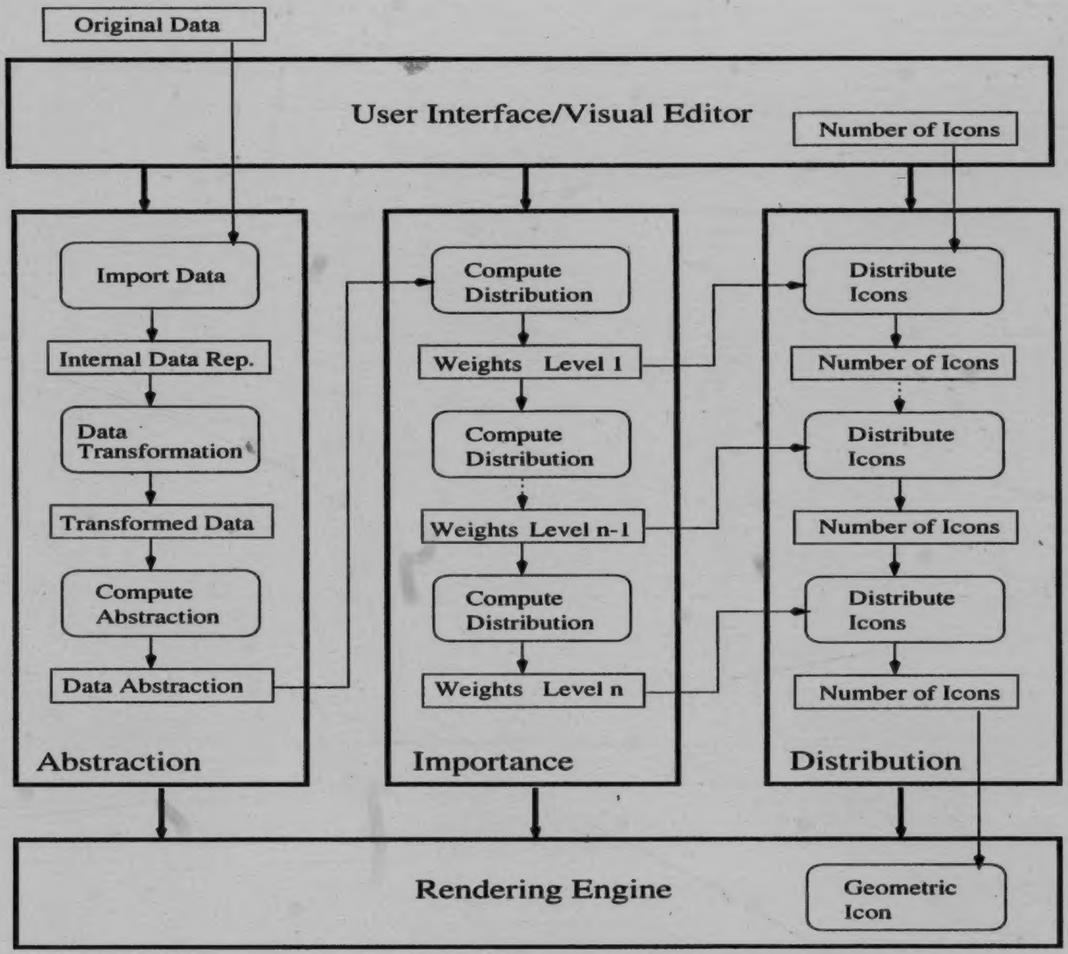
5.2 System Overview

Eikon consists of two parts: an object oriented toolkit (called *the Eikon Tool Kit* or *ETK*) and an associated visual programming environment. ETK, discussed in this chapter, consists of a number of classes and methods to create "visualization networks", that define the multi-step process of data transformation and display. The visual programming environment, discussed in the next chapter, supports a visual editor that simplifies the creation, configuration, and execution of visualization networks.

The visualization process is represented as an iterative process of transformation and display. ETK divides this process into three distinct stages: (1) a transformation and abstraction stage, (2) a multi-scale distribution stage, and (3) a data display stage. The transformation and abstraction stage performs the usual data transformations associated with the visualization process. ETK differs in that this stage is terminated by the computation of a *data abstraction* that determines the presence of features of interest in the data. The abstraction is used to compute a multi-scale distribution structure in the next stage, which in turn is used to determine the distribution of glyphs in the final stage. The overall architecture of the visualization subsystem is illustrated in Figure 5.1 and each stage is described in detail in the following sections.

5.2.1 Transformation and Abstraction

The goal of my system is to highlight interesting structures in the data by importing the original data into the system and applying a number of transformations to it. Data transformations in ETK are divided into two categories: those that filter the input data by extracting regions, or sub-sampling the data, and those that produce



The overall system architecture. The system is implemented as a number of data flow pipelines used to compute abstractions and to distribute glyphs. An image is produced by the selection of an abstraction function, a geometric glyph to display the data, and a number of glyphs to use. A multi-scale importance representation is computed from the abstraction which is used to distribute a selected number of glyphs throughout the image.

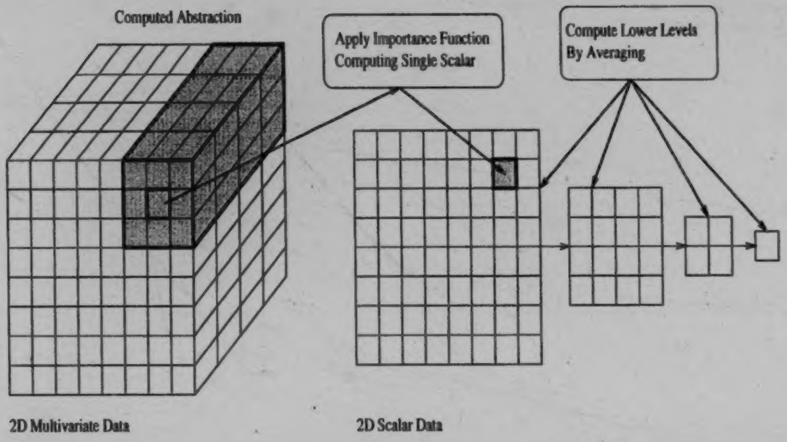
Figure 5.1: The system architecture of Eikon.

new data from the input data. In ETK, transformations that produce new data are called *abstraction functions.* The goal of the abstraction function is to compute a dataset that represents some feature of interest associated with the original data.

5.2.2 Multi-scale Importance and Distribution

When large amounts of data are displayed, the system must ensure that patterns and structures within the abstraction, or original data, remain visible. To accomplish this, an *importance function* is applied to the abstraction, or the original data, that computes a single scalar for each data point representing a defined important characteristic of it. The objective is to produce a simplified representation of the data, one that doesn't contain all information, but focuses on some important features of it. The importance representation is used to determine the location of a user-specified number of glyphs in the final image. There are a number of possible ways in which this can be done. The simplest technique is to threshold the importance values and assign glyphs to locations in the dataset that are above the threshold. This is the method used by Furnas [furn86]; however, it results in features in the dataset being overlooked. Thresholding also doesn't provide explicit control over the number of glyphs displayed. If all values are above the threshold then all data points will be displayed. Another method is to sort the importance values into descending order and to distribute glyphs starting with data with the highest values. Using the sorting technique, data points are assigned glyphs based on their importance value alone; the assignments are made irrespective of spatial location or scale. Ideally distribution should take into account the position and scale of features in the data.

ETK uses a different approach to glyph placement. Placement of glyphs is based on the magnitude of the importance values, their position, and scale. Glyph placement is determined by computing a multi-scale representation of the importance values (a quad-tree) and using this structure to distribute glyphs over the image. The importance quad tree is computed in a manner similar to mip-mapping [will83] by recursively averaging the initial output of the importance function as shown in Figure 5.2 and Equations 5.1 and 5.2. In this equation l is the level in the quad-tree, with $l = 0$ being the highest resolution importance image; $Importance_{l,x,y}$ denotes the importance value at level l and position (x, y); and (i, j) are offsets added to the



The importance process applies an importance function at every point in the dataset computing an importance representation. The importance representation consists of a single scalar indicating the response to the importance function at each point. A multi-scale importance pyramid (a quad tree) is computed from this representation by recursively averaging the initial importance representation

Figure 5.2: Computing the multi-scale importance representations.

current location to compute the average. The importance quad-tree is computed by recursively applying the averaging function until a single importance value is obtained.

$$Total_{l+1,x,y} = \sum_{i=0}^{i=1} \sum_{j=0}^{j=1} Importance_{l,(x\times 2)+i,(y\times 2)+j} \tag{5.1}$$

$$Importance_{l+1,x,y} = \frac{Total_{l+1,x,y}}{4.0} \tag{5.2}$$

Distribution Weights

The next phase of the process involves the computation of distribution weights for all importance values in the quad tree. The distribution weights determine the percentage of available glyphs to distribute to the region of the data associated with each importance value. By default the root node of the quad tree (representing the entire dataset) receives an importance value of 1.0. This simply means that all the user specified number of glyphs should be distributed over the dataset. All nodes in the quad tree (except those at the lowest level of the tree) have four child nodes and associated importance values. As illustrated in Equations 5.3 - 5.6, the distribution weight assigned to a particular child is based on this child's contribution to the total sum of importance values for all four children.

$$Weight_{l,(x\times2),(y\times2)} = \frac{Total_{l+1,x,y}}{Importance_{l,(x\times2),(y\times2)}} \tag{5.3}$$

$$Weight_{l,(x\times2)+1,(y\times2)} = \frac{Total_{l+1,x,y}}{Importance_{l,(x\times2)+1,(y\times2)}} \tag{5.4}$$

$$Weight_{l,(x\times2)+1,(y\times2)+1} = \frac{Total_{l+1,x,y}}{Importance_{l,(x\times2)+1,(y\times2)+1}} \tag{5.5}$$

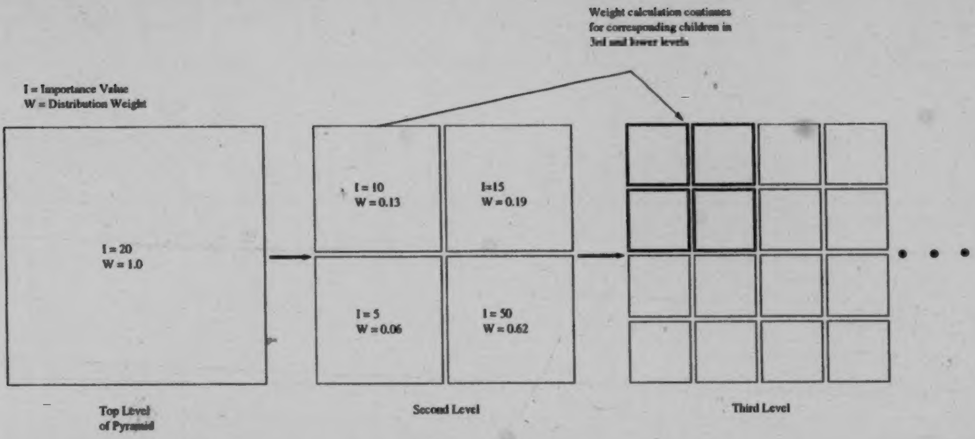
$$Weight_{l,(x\times2),(y\times2)+1} = \frac{Total_{l+1,x,y}}{Importance_{l,(x\times2),(y\times2)+1}} \tag{5.6}$$

Equations 5.3 - 5.6 indicate that the weight of node (x, y) at level l is determined by the sum of this node and its siblings importance values divided by the importance value assigned to the node (x, y). This process is also illustrated in Figure 5.3.

Glyph Distribution

Before distribution begins, the user specifies a number of glyphs available to distribute in the image. This number is assigned to the root node in the importance quad-tree. From this, the number of glyphs to distributed over an area defined by a node at a lower level (l) is determined by the number available at its parent (level $l + 1$) scaled by the weight of the node (Equations 5.7 - 5.10.) As only an integer number of glyphs can be distributed the value assigned to the node is the truncated value of glyphs available scaled by the weight. This may result in some glyphs not being initially distributed due to truncation error (Equation 5.11.) In this case the remaining glyphs are distributed to areas with the largest truncation error (Equations 5.12 - 5.15.)

$$\begin{gathered} Distributed_{l,(x\times2),(y\times2)} = \\ Trunc(Available_{l+1,x,y} \times Weight_{l,(x\times2),(y\times2)}) \end{gathered} \tag{5.7}$$



The top level element has an importance value of 20 and is assigned a weighting of 1.0 by default. The elements in the second level of the pyramid have importance values 10, 15, 5, and 50 as shown. Importance weights for each element are computed based on their contribution to the sum of these four values, giving values of $\frac{10}{80}(0.13)$, $\frac{15}{80}(0.19)$, $\frac{5}{80}(0.06)$, and $\frac{50}{80}(0.62)$. The process of importance weight calculation then proceeds to the children of each element.

Figure 5.3: Computing importance weights for glyph distribution.

$$\begin{aligned} &Distributed_{l,(x\times2)+1,(y\times2)} = \\ &Trunc(Available_{l+1,x,y} \times Weight_{l,(x\times2)+1,(y\times2)}) \end{aligned} \tag{5.8}$$

$$\begin{aligned} &Distributed_{l,(x\times2)+1,(y\times2)+1} = \\ &Trunc(Available_{l+1,x,y} \times Weight_{l,(x\times2)+1,(y\times2)+1}) \end{aligned} \tag{5.9}$$

$$\begin{aligned} &Distributed_{l,(x\times2),(y\times2)+1} = \\ &Trunc(Available_{l+1,x,y} \times Weight_{l,(x\times2),(y\times2)+1}) \end{aligned} \tag{5.10}$$

$$Remainder_{l+1,x,y} = Available_{l+1,x,y} - \left(\sum_{i=0}^{i=1}\sum_{j=0}^{j=1} Distributed_{l,(x\times2)+i,(y\times2)+j}\right) \tag{5.11}$$

$$\begin{aligned} Error_{l,(x\times2),(y\times2)} = {} & (Weight_{l,(x\times2),(y\times2)} \times Available_{l+1,x,y}) \\ & - Distributed_{l,(x\times2),(y\times2)} \end{aligned} \tag{5.12}$$

$$\begin{aligned} Error_{l,(x\times2)+1,(y\times2)} = {} & (Weight_{l,(x\times2)+1,(y\times2)} \times Available_{l+1,x,y}) \\ & - Distributed_{l,(x\times2)+1,(y\times2)} \end{aligned} \tag{5.13}$$

$$
\begin{aligned}
Error_{l,(x\times2)+1,(y\times2)+1} = (&Weight_{l,(x\times2)+1,(y\times2)+1} \times Available_{l+1,x,y}) \\
&-Distributed_{l,(x\times2)+1,(y\times2)+1}
\end{aligned}
\tag{5.14}
$$

$$
\begin{aligned}
Error_{l,(x\times2),(y\times2)+1} = (&Weight_{l,(x\times2),(y\times2)+1} \times Available_{l+1,x,y}) \\
&-Distributed_{l,(x\times2),(y\times2)+1}
\end{aligned}
\tag{5.15}
$$

Referring to Figure 5.3, if we assume 50 glyphs have to be distributed then the second level elements receive $(0.13 \times 50) = 6.5, (0.19 \times 50) = 9.5, (0.06 \times 50) = 3$, and $(0.62 \times 50) = 31$ glyphs. Each value is truncated resulting in one glyph not being distributed. In this example two elements have the same error of 0.5, so we either distribute the remaining glyph randomly between the two or in some other defined order.

The distribution process continues through the levels of the pyramid until either the bottom of the pyramid is reached or no more glyphs remain to be distributed. If no more glyphs remain to be distributed the data in the associated region is not of interest given the current abstraction function, importance function, and number of glyphs. If the lowest level of the pyramid is reached up to four glyphs will be available for display. These glyphs are placed starting with the element with the largest importance value and proceeding until all glyphs have been placed. Again if elements have the same importance value the glyphs are distributed randomly or in some predefined order.

5.2.3 Data display

Once all glyphs have been distributed data points will be displayed at the x and y positions indicated by the distribution. The data points are displayed by binding their data variables to the parameters of a user-selected glyph. This results in an image that has a number of interesting properties. Glyphs are placed such that interesting areas receive relatively more glyphs than uninteresting areas. As a result, when there is only a small space for each glyph, i.e., when the ratio of pixels to data points is low, the user is presented with an overview of the data through variation in glyph density. The user can select and explore these areas by changing the abstraction function, the glyph, the data-glyph binding, or the number of glyphs used. Exploration of a local area results in fewer data points and therefore a higher pixel to data ratio. In this instance, spatial density can also be used to render important structures visible and the increased pixel to data ratio increases the visible geometric variation within individual glyphs. Finally when only a few glyphs are displayed on screen, there is a large number of pixels to display each glyph, and therefore the supported geometric variation allows relatively precise readings of data variables on an elementary level.

5.3 System Design and Implementation

The goal of ETK is not only to support the described process of data visualization but also to provide a flexible and extensible system. The ability easily to extend[1] the system is important because the process of data visualization results in users thinking about other aspects of the data and creates the desire to view the data using different abstraction functions or data glyphs. It is impossible for our prototype system (or any system) to incorporate all possible abstraction functions or glyphs or even enumerate all possible ways of using them. Therefore an important design goal has been the development of an architecture that allows easy construction of multiple visualizations and easy addition of new system functionality as needed.

Pipes and Filters

As a result of the "pipelined" nature of the visualization process, a natural system pattern for visualization systems has been the "Pipes and Filters" pattern [busc99]. Within this architecture, processes and the data on which they operate on are represented as distinct entities. Although somewhat different from traditional object oriented approaches, where data and processes (methods) are incorporated into a single object, this approach is used almost universally throughout production visualization systems [upso89, luca92, kons94, schr98]. Some research into more traditional object oriented design methods, where classes encapsulate both data and processes [favr94] has demonstrated that this approach can produce novel ways of generating visualizations also. Modeling processes and data separately conforms more closely to users' conceptual model of visualization, where both data and process are regarded as related but distinct entities [schr98]. The modularity afforded by representing process and data in this way also simplifies visualization development and maintenance. This approach allows the system to be easily extended by adding new abstraction functions, new importance functions, or new data glyphs. This method however requires that the data representation and the interface between data and process be carefully designed. A poorly designed data representation can introduce large overheads in data storage and can cause problems with connecting processes together through the output/input they use. A poor data interface introduces problems of efficiency in accessing and modifying data.

---

[1] Easy for a visualization expert to extend and not necessarily a visualization user.

5.3.1 Data Representation

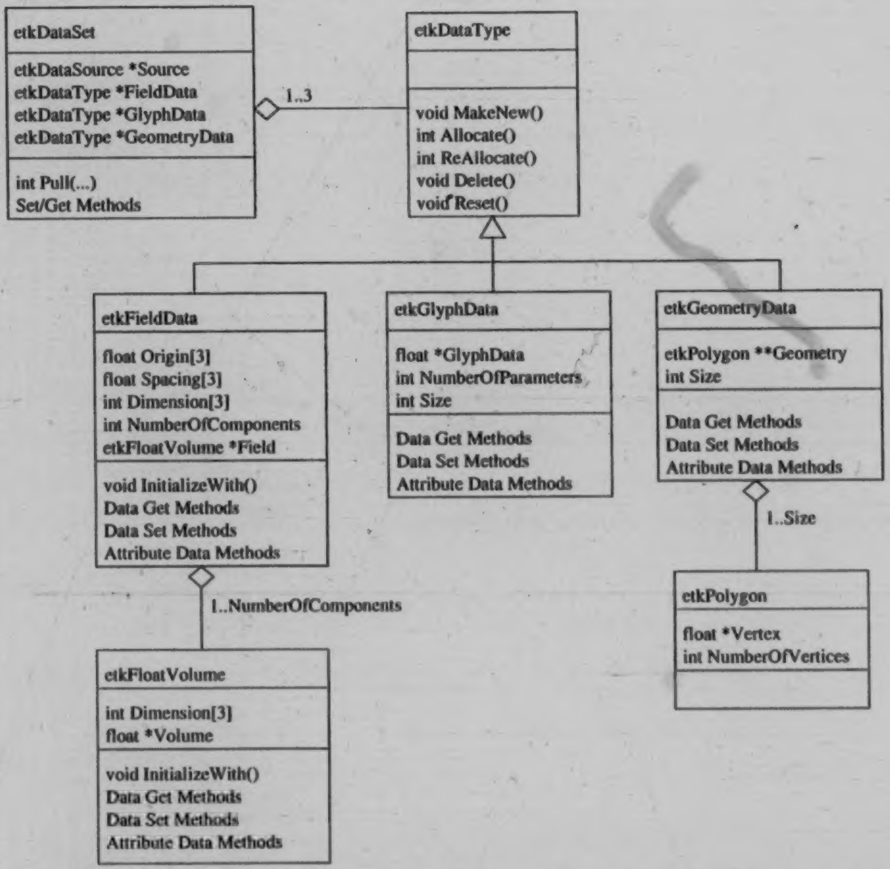
ETK focuses on regular two-dimensional spatially referenced multivariate data (often referred to as structured points[2].) As a result we have chosen to use a single class, the `etkDataSet` class to represent all data objects within the system. A schematic of this class is shown in Figure 5.4.

The use of a single data object simplifies the connection between process objects and data objects: each process object simply expects a `etkDataSet` object as input. Visualization networks are constructed by connecting process objects together. A process object that generates one type of data can only be connected to another process object that consumes the same type of data, therefore using a single data object can simplify network construction. The primary disadvantage of this choice is that all processes must be designed to work on a common data representation: this can introduce inefficiencies in computation when a process object can't access the data it requires easily and inefficiencies in representation when new data is encountered that doesn't fit well with the scheme used.

In ETK the affects of these problems are reduced by defining the `etkDataSet` class as an aggregate of three sub-classes, reflecting the three major classification of data: `etkFieldData`, used to represent multivariate data volumes; `etkGlyphData`, use to represent data associated with groups of glyphs; and `etkGeometryData` used,

---

[2] Currently the system is used to visualize two-dimensional spatial data. The data structures actually presented here support three-dimensional data also.



A dataset is represented as a collection of between one and three data types. Data types in turn can represent either field data, glyph data or geometry data. Field data is defined as an aggregation of floating point volumes; glyph data describes the various properties of glyphs; and geometric data is defined as a collection of three-dimensional polygons

Figure 5.4: The etkDataSet and etkDataType class hierarchy.

to represent traditional polygonal data.

Field Data

Field data consists simply of a collection of floating point arrays. Each array represents a single variable of the data, and the dimensionality of the array is determined by the spatial dimensions of the dataset. Each floating point array is represented by `etkFloatVolume`, that contains an allocated block of floating point numbers, one for each data point. The float volume class also supports several methods to retrieve and modify data within the volume. Data can be retrieved and modified based on single points, rows, slices, or the entire volume depending on the needs of specific algorithms, providing fast and efficient access to the data.

Glyph Data

Glyph data is used to represent the different values for each of the variables associated with a group of glyphs. For example a line segment glyph has glyph data corresponding to the x, y, and z positions of the glyph, the dx, dy, and dz values for the line direction and magnitude, a value for the size of the bulb associated with the glyph , and a value for the glyph color. This data is represented by a one-dimensional array of floating point values (see `etkGlyphData` as shown in Figure 5.4) within the `etkGlyphData` class. The size of the array is governed by the number of parameters associated with each glyph and the number of glyphs present. Again the glyph data class supports methods for accessing individual glyph parameters, individual glyphs and groups of glyphs.

Geometry Data

The `etkGeometryData` class represents geometric data in the form of collections of polygons. Polygons are represented via a simple `etkPolygon` class that defines polygons as arrays of floating point vertices. The use of geometry data is necessary to support rendering of traditional polygonal models. Such models are useful for providing a context for the data by rendering the outline of the displayed dataset for example. This class allows access to individual polygons or groups of polygons if needed.

Type Checking

Although the use of three sub-classes for specific types of data allows more efficient storage of particular types of data generated by the system, it introduces the need to perform some form of type checking. Process objects can accept the `etkDataSet` class as input but may only be interested in one of the three specific data sub-classes. For example the `etkGeometryMapper` process expects geometric data containing polygons as input; if this type of data doesn't appear then the process can't execute properly. In ETK, type checking is performed internally within each process object at run time. This involves checking to see if the appropriate data is present and if not returning an error message to indicate the problem. Additional type checking is supported at "programming" time within the visual programming environment discussed in the next chapter.

5.3.2 Dataset Interface

The `etkDataSet` class defines an interface that allows access to each of the data types, supports connectivity in the network, and defines the memory management protocol in use.

- Creating, accessing and setting data

  Setting and retrieving the data types is defined through `Set` and `Get` functions that retrieve and set pointers to the three types of data that the dataset may contain. Access and modification of the actual data is delegated to each of the three sub-classes, where they implement the `etkDataType` interface. This interface describes how each data type is constructed, how it is deleted, how space is allocated, and how the data is initialized. It is an abstract class and concrete implementation of it is provided by the sub-classes as shown in Figure 5.4.

- Connectivity

  A data object must know which source object created it and must tell the source object when it may be necessary to generate new data. [3] The data/source connection is defined by maintaining a pointer to the source process in the dataset object (`etkDataSource *Source`). When data is requested the dataset object uses this pointer to send the request to the source object associated with it. The source object will in turn populate the dataset object with new data if necessary.

---

[3] This is due to the control mechanism used in Eikon which is discussed later

- Memory management

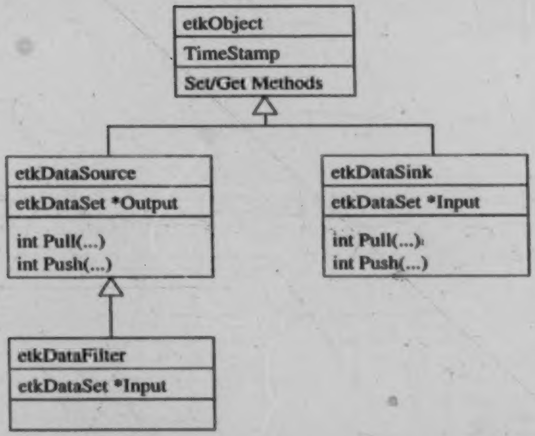
  Memory management is important to make efficient use of both the memory and computation resources available. Therefore `etkDataSet` defines an interface that allows different memory management protocols to be used to manage the memory associated with the data. Eikon currently supports a simple memory management protocol (similar to that used in the Visualization Toolkit (VTK) [schr98]), that allows process objects to store the data when it is expensive to recompute or to delete the data when memory resources are low.

5.3.3 Process Representation

Processes are responsible for importing data into the system, transforming data at various stages in the pipeline, computing abstractions and importance representations, and finally rendering the data in an image or writing the data to a file. As in other visualization systems, processes can be categorized as either *Data Sources*, which are producers of data, *Data Filters*, which transform data or *Data Sinks*, which are the final consumers of data in the visualization. Data sources, filters, and sinks can be determined by the number of inputs they use and the number of outputs they generate. Sources have no inputs and generate at least one output, filters have at least one input and generate at least one output, and sinks have at least one input and generate no output, or generate output that is external to the system. The relationship between each type of process is shown in Figure 5.5.

Data Sources

Data sources are the interface between data that exist outside the system and the representation of this data within the system. The base class for data sources is the `etkDataSource` class illustrated in Figure 5.6. The data source object must



Processes in ETK are represented as either data sources, data filters, or data sinks. Data sources are net generators of data, data filters transform the data as it moves through the pipeline, and data sinks are the net consumers of data.

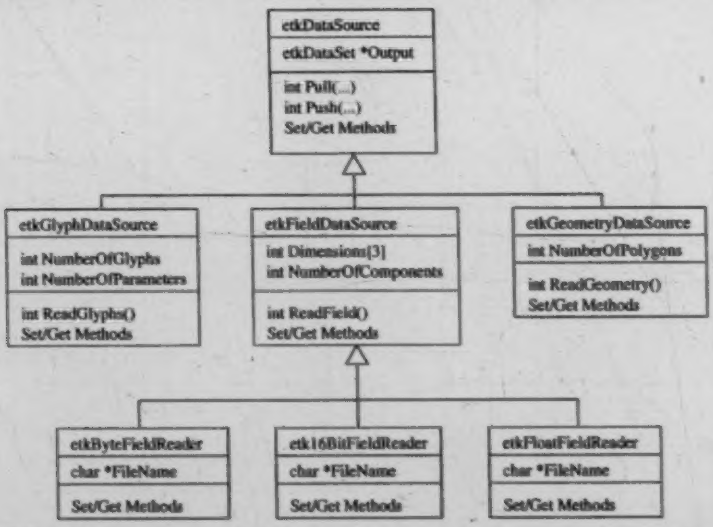
Figure 5.5: Process representation in ETK.

create an output corresponding to an instance of the `etkDataSet` object that will contain one or more of the three basic data types `etkFieldData`, `etkGeometryData`, or `etkGlyphData`. The source of the created dataset is established by setting the `Source` pointer in the `etkDataSet` object.

The base `etkDataSource` class delegates the actual creation of output data types to three sub-classes representing each type: `etkFieldDataSource`, which supports reading field data; `etkGlyphDataSource`, which supports reading glyph data; and `etkGeometryDataSource`, which supports reading geometry data. Addition of data source objects is a straightforward process requiring only new sub-classes of the data sources classes with the necessary methods overridden. ETK contains three examples of such concrete classes corresponding to field data sources that process 8 bit, 16 bit, and floating point datasets.

Data Filters

The base class for all data transformations and abstractions is the `etkDataFilter` class (Figure 5.7). Because the major difference between a filter and a source is that the filter also has an input dataset as well as an output dataset the filter class itself is a sub-class of `etkDataSource`. Network connections are formed by using the output



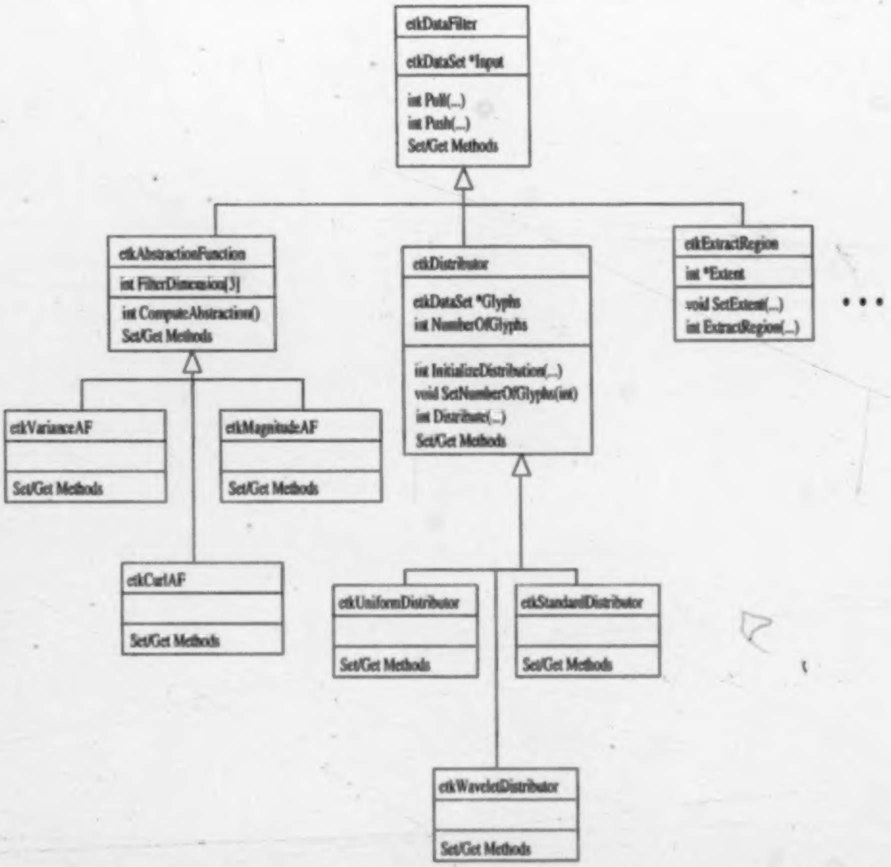
ETK defines three basic types of data sources corresponding to three basic types of data. Each type of data source defines an interface that concrete versions of the data source must implement. Other data sources can be created by sub-classing from the data source object.

Figure 5.6: The etkDataSource class hierarchy.

of upstream data sources (or data filters) as the input to downstream data filters (or sinks). Data sources define a `GetOutput()` method that returns a pointer to the process's output and data filters define a `SetInput()` method that defines which dataset forms the input to the filter.

New filters can be added to ETK by sub-classing the **`etkDataFilter`** class and overriding the necessary methods to enable the transformation of input data to the output data. The base filter class contains a single input and a single output. Subclasses of this, however, can define other inputs and outputs as needed. For example, a re-sample filter may provide extra input defining the spatial location of sample points.

Within ETK, two classes of data filters are of special significance: abstraction and importance functions and glyph distribution filters.



Within the range of data filters defined in ETK, two types have special significance: the abstraction function, and the distributor. These abstract classes define an interface that concrete implementations of abstractions and distributors must implement.

Figure 5.7: The etkDataFilter class hierarchy.

Abstraction and Importance functions

The `etkAbstractionFunction` class defines how abstraction and importance functions are implemented. This class is a subclass of the `etkDataFilter` class and its interface stipulates that all concrete abstraction functions must implement a `ComputeAbstraction()` method. This method performs the necessary abstraction on the input data. ETK doesn't provide an explicit class for importance functions. These can be easily modeled as abstractions that produce datasets containing a single scalar for each data point.

Distributors

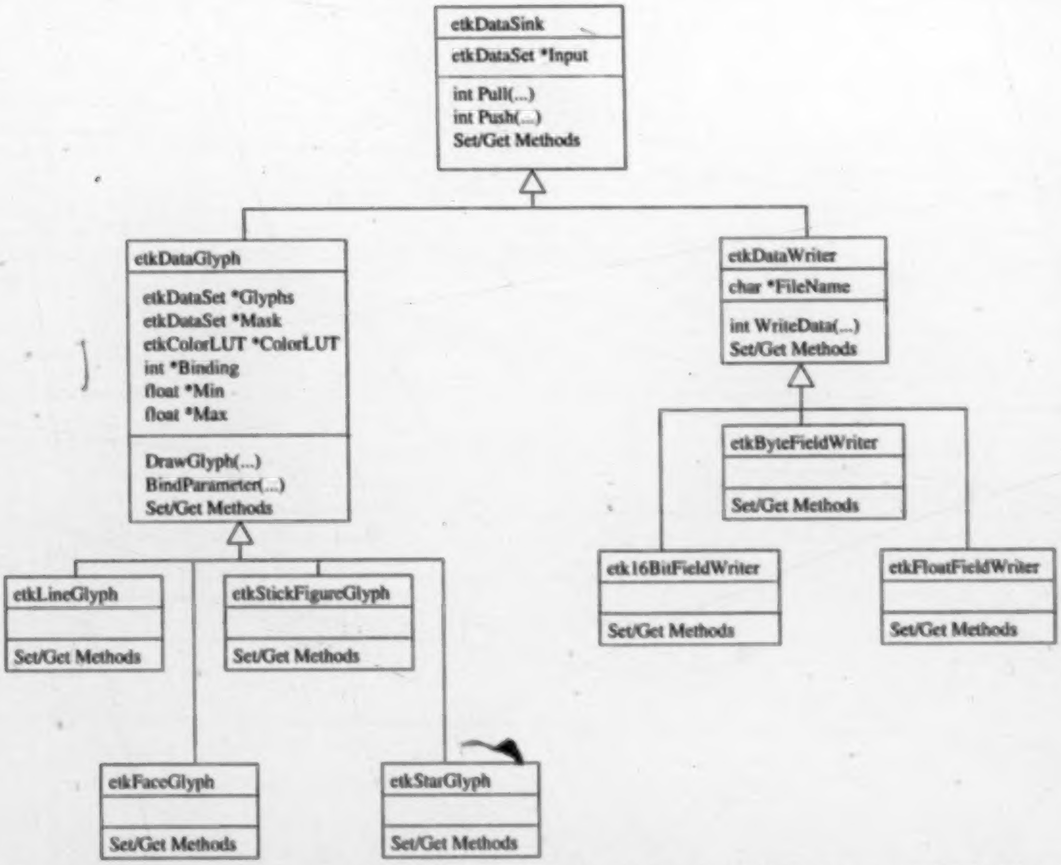
Distributors are responsible for computing the distribution of glyphs over the image. The distribution process is defined by the `etkDistributor` class, an abstract class defining the interface that concrete distributors must implement. Each distributor uses two input datasets, corresponding to the importance values and the number of glyphs to distribute. The exact nature of the distribution is governed by the implementation of the `Distribute()` method. Various distribution methods can be supported by providing different implementations for this method. ETK currently supports two types of distributors.

1. Uniform Distribution

   The uniform distributor (`etkUniformDistributor`) assumes an equal weighting for each area irrespective of the abstraction or importance function used. This results in memory and computation savings since there is no need to compute the multi-scale distribution pyramid. The distribution is computed by subdividing the spatial extent of the data into a number of subregions depending on the number of glyphs to be displayed. The data is displayed by associating a glyph with the data point at the center of each region.

2. Standard Distribution

   The standard distributor (`etkStandardDistributor`) implements the distribution process described in section 5.2.2 by distributing glyphs based on a multi-scale representation of importance values. The standard distributor computes



Two abstract data sinks are defined: the data glyph for rendering data on the display, and the data writer for storing data back on disk. Each of these defines an interface that concrete versions of the class must implement.

Figure 5.8: The etkDataSink class hierarchy.

levels in the pyramid as they are required and frees the memory associated with them when they are no longer needed. This reduces the memory needed to store a complete importance pyramid at a cost of recomputing the levels each time a different number of glyphs are distributed.

Data Sinks

Data sinks represent the termination of data flow within the visualization network. As shown in Figure 5.8, ETK defines two types of data sinks: `etkDataGlyphs` and `etkDataWriters`.

Data Glyphs

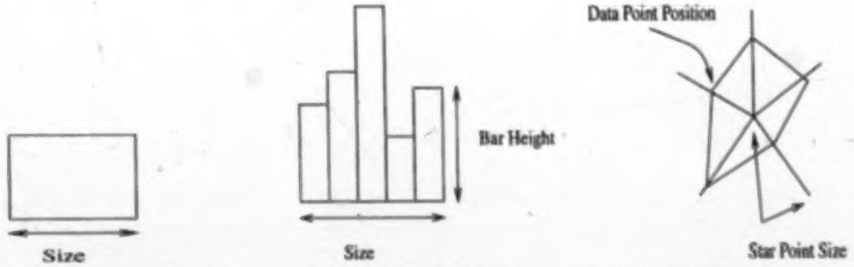
Data glyphs have three inputs: the `Input`, which corresponds to the multivariate data that will be represented by the glyphs; the `Mask`, which determines the spatial location of glyphs; and the `Glyphs`, which contains a series of "pre-computed" glyphs that should be displayed. The presence or absence of these inputs governs the behavior of the data glyph object. If `Input` is present, then the data it contains is used to draw glyphs by placing a glyph at every data point. If a `Mask` is also present then glyphs are place selectively at locations determined by the mask. If only `Glyphs` are present this data is used to draw glyphs directly. If both `Input` and `Glyphs` are present then the `Input` is used to draw glyphs and the `Glyphs` are ignored.

The `etkDataGlyph` class also defines a number of internal variables that describe the glyph being rendered. These correspond to the number of parameters that can be used to represent the data (excluding the three positional parameters of the glyph); a `Binding` array, that indicates which data variables are bound to which glyph parameters; a minimum and maximum value for each glyph parameter; and normalization factors, which are used to ensure that the range of the data values fall within the minimum and maximum range associated with each parameter.

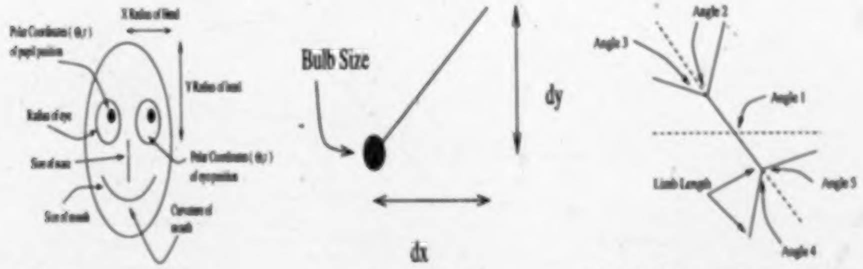
As usual, the `etkDataGlyph` class is an abstract class from which concrete classes are derived. Concrete glyphs are defined by overriding the `DrawGlyph()` method of the base class. This method performs the actual rendering of the glyph on the screen based on the supplied input data and binding array. The world coordinate position of the glyph is computed using the position of the dataset's origin, the i and j index of the data point in the dataset, and the inter-data point spacing. If no input is present the glyph data contained in `Glyphs` is used to generate the display. In each case, the glyph is drawn by passing all glyph parameters to the `DrawGlyph()` method.

Implemented Glyphs

The glyphs currently implemented in Eikon are shown in Figure 5.9 (reproduced here from Chapter 3). These correspond to a cross section of the geometric coding methods discussed in Chapter 3 and where chosen because they allow us to highlight some of the advantages and disadvantages of geometric coding of large multivariate datasets



(a) Square Glyph (b) Profile Glyph (c) Star Glyph



(d) Face Glyph (e) Line Glyph (f) Stick Figure Glyph

Figure 5.9: The glyphs implemented in Eikon.

using the methods incorporated into Eikon.

All glyphs share common parameters: three describing x, y and z position, one describing size and one describing color. Each glyph also supports a varying number of other parameters to represent data variables.

- Square glyph

  The most basic glyph in Eikon is the square glyph [ware88b] which represents data by spatial position, size, and color.

- Profile Glyph

  The profile glyph [bert83] is a simple extension to the square glyph which uses a number of vertical bars to represent data variables. The profile glyph implemented uses five bars and allows other data variables to be represented through the overall size and color of the glyph.

- Star Glyph

The implemented star glyph [sieg71] uses a five pointed star to represent each data point. Data variables are represented by distances along each star point, the overall size of the star, and its color.

- Face Glyph

  The face glyph is a simplified version of the Chernoff face glyph [cher73] and allows data variables to be represented through the x and y radius of the face, the r and θ polar coordinates of the eye position, the radius of the eye, the r and θ coordinates of the pupils, the size of the nose, the size, of the mouth, the curvature of the mouth, the color of the face, and the overall size of the face. This provides the possibility of representing up to twelve different data variables, not including spatial position, in a single glyph.

- Line Glyph

  The line glyph [hall86] is implemented using a small line segment and represents data variation through the dx, dy and dz coordinates of the line slope, the size of the bulb, and the color of the glyph.

- Stick Figure Glyph

  The final glyph implemented is the stick figure glyph [pick88]. The implemented stick figure glyph consists of two pairs of limbs connected to a main body. Data variation is represented via the slope of each limb and the main body. The length of each limb and the color of the stick figure can also be used, giving the possibility of representing a maximum of seven data variables not including spatial position.

Adding New glyphs

These glyphs are only a sample of the many possible glyph designs. As new glyphs are designed they can be added to the system by creating new sub-classes of the `etkDataGlyph` class. These sub-classes must provide the number of parameters the glyph supports, default minimum and maximum values for each parameters, and define the rendering method to display the glyph. The base glyph class is responsible for all other processing, including the binding between data variable and glyph parameters and the normalization of data values.

Data Writers

The `etkDataWriter` class allows data stored internally within the system to be committed to secondary storage. Several sub-classes of `etkDataWriter` allow different data types to be stored in a variety of formats. The data writers allow data to be written to a file or to standard output. Writing to standard output allows the results to be "piped" into other applications. In this way `etkDataWriter` objects can export the ETK data to be viewed or used by external applications. Again, `etkDataWriter` is an abstract class that defines the interface for data writers. Concrete versions of this class are defined by providing implementations of the `int WriteData()` method.

5.4 Execution and Control

To use the process and data objects to construct networks and produce visualizations, an execution and control policy must be defined. Execution and control ensure that processes are executed if they are needed. Processes only need to execute if their inputs or internal variables have changed since the process last executed and the output they generate is required. Execution and control can be accomplished in two ways: via centralized control or via distributed control.

5.4.1 Centralized Control

The centralized control method is used in many visualization systems including AVS [upso89] and IBM Data Explorer [luca92]. This method uses a central control object to maintain changes to variables and processes within the network. When output is required, the central control determines dependencies between connected processes and uses this to determine which processes need to execute.

The primary advantage of this method is that analyzing the topology and dependencies within the visualization network allow for memory and computation optimizations. The main disadvantage of this method is that it is difficult to implement and in addition, each process is inherently linked to the central control object. Also computing dependencies between processes is often not possible, given that some processes may conditionally execute based on the values of local variables.

5.4.2 Distributed Control

The distributed control method, used in toolkits such as VTK [schr98], delegates the responsibility of execution and control to each of the processes in the network. Under this scheme there is no notion of global topology; processes are indirectly aware of their immediate predecessor and ancestor processes via their input and output datasets. Execution and control is supported by propagating requests for data from data sinks to data sources and by sending data generated from the data sources back towards the data sinks. Processes determine if they should generate data by performing a consistency check that compares their last execution time against the modification time of their inputs and internal variables. Distributed execution and control methods are simple to implement, and require no global knowledge of network topology. Distributed execution requires tracking the modification and execution time of each process. Processes are not concerned directly with how their inputs are generated or how their outputs are consumed, but only if their inputs have been modified or if their output exists. The main disadvantage of this scheme is that several inefficiencies in execution can arise without knowledge of global topology. Error reporting and recovery are more difficult also. Finally, debugging of complex and interconnected networks can also be difficult.

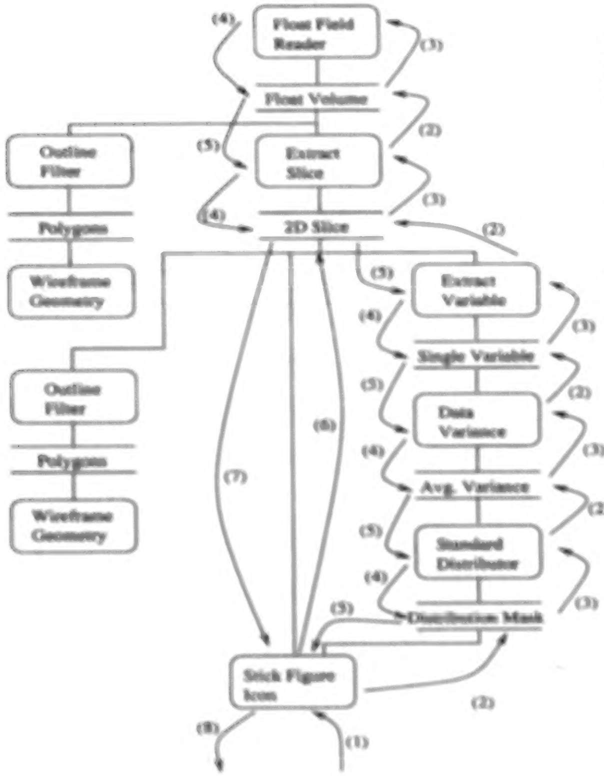
5.4.3 Execution and Control in Eikon

Eikon uses the distributed execution and control model. In a manner similar to VTK, ETK implements the distributed control scheme through the use of `Pull()` and `Push()`[4] methods associated with each process object. The `Pull()` method "pulls" data from upstream objects, while the `Push()` method generates new data and "pushes" it to downstream objects. Control of the pull/push cycle is achieved by using modification and execution times associated with process and data objects. Modification times are updated whenever a variable associated with the process or data object is changed, and execution times are updated whenever a process object generates new data. The push/pull cycle is initiated by a request for data (pull request) being made to a data sink in the visualization network. This request is propagated back toward the relevant data sinks through links defined between the data sink, upstream data, and process objects. Once a data source object is encountered, this object determines whether to generate new data by comparing its modification time against its last execution time. If it is not up-to-date the process must re-execute and push the new data downstream. At this point, control returns to the downstream objects which perform the same consistency check on their internal variables and data inputs. The pull/push cycle ends when the data sinks receive and process the data they need. The basic algorithm for execution and control is shown below:

---

[4] VTK uses *Update()* and *Execute()* methods instead

```
For each required INPUT
{
   if INPUT not present then
   {
      Display an error
      Return
   }
}
if PULL is true
{
   Return as we are in a "pull" loop
}
Set PULL to true
Set REGENERATE to false
For each INPUT
{
   if present then
   {
       Send Pull request to INPUT
       if INPUT modification is newer than PROCESS execution then
       {
          set REGENERATE to true as we are now out of date.
       }
   }
}
Set PULL to false
if PROCESS modification time is newer than PROCESS execution time
{
   set REGENERATE to true
}
if REGENERATE is true
```



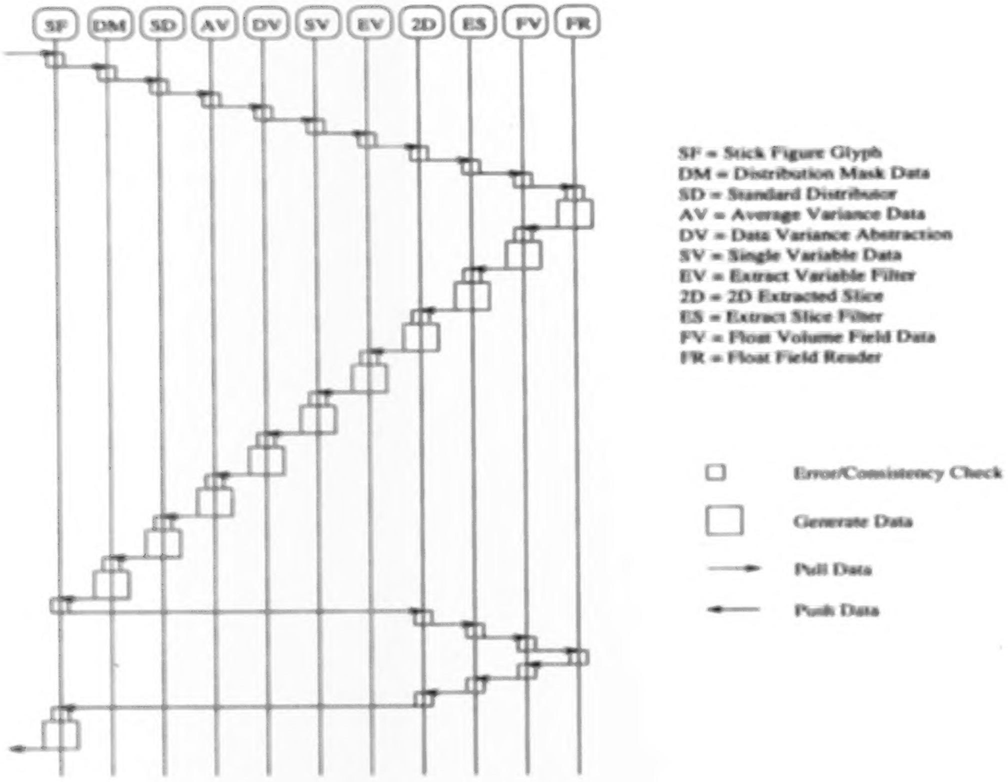
(1) Initial Pull request for data
(2) Pull request is propagated to input
(3) Pull request propagated to source of input
(4) Pull request reaches data source. The data source generates the data and pushes it downstream
(5) Control returns to downstream objects which in turn generate and push data as in (4)
(6) Pull request is sent to second input which is propagated back to source as in (2)/(3). This time processes realize data is still current and push existing data downstream.
(7) Control returns to the data sink
(8) The data sink renders an image on the screen

The numbered arrows show the direction and sequence of events that are explained by the table on the right. An initial request for data is propagated back towards the data sources through the network connections defined by the data objects. At the data source data is generated, it is sent back down the network, being transformed as needed by each process object.

Figure 5.10: The Pull/Push pattern through a network in ETK.

```
    {
        Regenerate output data
        Push output data down stream
        Update PROCESS execution time.
    }
```

A detailed example of how this pull and push mechanism operates within ETK is shown in Figures 5.10 and 5.11. Figure 5.10 shows the collaboration between processes and data during the push/pull operation, while Figure 5.11 shows the sequence of execution associated with the different processes in the network.


SF
DM
SD
AV
DV
SV
EV
2D
ES
FV
FR
SF = Stick Figure Glyph
DM = Distribution Mask Data
SD = Standard Distributor
AV = Average Variance Data
DV = Data Variance Abstraction
SV = Single Variable Data
EV = Extract Variable Filter
2D = 2D Extracted Slice
ES = Extract Slice Filter
FV = Float Volume Field Data
FR = Float Field Reader
Error/Consistency Check
Generate Data
Pull Data
Push Data

The sequence of pull/push calls and associated execution patterns for the network are described in this image. This shows the slight overhead involved with the second pull request originating with the stick figure glyph. This involves consistency checks at each downstream process but no re-execution occurs in this example. Each process object maintains its output so the output is current. If the output was released then this phase would require the downstream objects to be re-executed.

Figure 5.11: The function call and execution pattern of a Pull/Push cycle.

In this example, the display process begins at the bottom with a request for output being made to the glyph objects. To render glyphs the Glyph process asks the Distributor where glyphs should be placed, and also asks the Slice object to provide data for the glyphs. In this respect the Glyph process is an example of an object with multiple inputs. This process of demanding output proceeds with the Distributor asking the Variable Extractor for the variable, the Variable Extractor requesting the data from the Slice Extractor, and the Slice Extractor requesting data from the Data Reader. The Data Reader corresponds to a data source; because its output doesn't exist, it must generate the data required by the downstream objects. At this point, the procedure unwinds with processes executing if necessary and pushing a series of datasets through the pipeline towards the glyph process, which renders it based on the computed distribution mask and extracted slice.

5.5 Example Code

I end this chapter with an example of a simple visualization network created to display the results of an Aurora simulation conducted by the Department of Physics at the University of Alberta. The dataset[5] being displayed consists of a 1024×1024×3 volume where a three-dimensional vector is defined at each point. The visualization involves the extraction of a 1024×1024 slice from this volume and the use of a line glyph to display each data point. A code snippet that shows the creation of the network as a connection of several process objects is shown below. This example illustrates how transformation, importance, distribution, and display are achieved. It should be noted that no abstraction function is applied in this example as we are interested in important characteristics of the original data, namely the vector magnitude.

```
void RenderData()
{
    // Create a float field reader, specify filename and dimensions

    etkFloatFieldReader *pReader = etkFloatFieldReader::MakeNew();
    pReader->SetDimensions(1024,1024,3);
    pReader->SetNumberOfComponents(3);
```

---

[5] The data was provided by Igor Voronkov, a research associate in the Space Physics Group

```
 8      pReader->SetFileName("./auroratest.etk");
 9
10      // Extract the middle slice from the generated data
11
12      etkExtractRegion *pSlice = etkExtractRegion::MakeNew();
13      pSlice->SetInput(pReader->GetOutput());
14      pSlice->SetExtent(0,1024,0,1024,1,1);
15
16      // Compute the magnitude of each vector as the importance
17
18      etkMagnitudeAF *pAbstractor = etkMagnitudeAF::MakeNew();
19      pAbstractor->SetInput(pSlice->GetOutput());
20
21      // Use the standard distributor to distribute using magnitude
22
23      etkStandardDistributor *pDistributor = etkDistributor::MakeNew();
24      pDistributor->SetInput(pReader->GetOutput());
25      pDistributor->SetNumberOfIcons(14000);
26
27      // Bind the data to a line glyph using the slice data and the
28      // computed distribution
29
30      etkLineGlyph *pGlyph = etkLineGlyph::MakeNew();
31      pGlyph->SetInput(pSlice->GetOutput());
32      pGlyph->SetMask(pDistributor->GetOutput());
33      pGlyph->BindParameter(0,0);
34      pGlyph->BindParameter(1,1);
35      pGlyph->BindParameter(2,2);
36      pGlyph->SetColor(0.0,0.0,1.0);
37
38      // Add the line glyph to the renderer and display the data.
39
40      pRenderer->AddObject(pGlyph);
41      pRenderer->Display();
42  }
```

In this example, an `etkFloatFieldReader` object reads the volume dataset into memory. The filename and dimensions of this object are set and default values of origin and spacing are used (lines 4-8). The output of the field reader is then given to the `etkExtractRegion` filter, which extracts the middle slice in the XY plane from the dataset (lines 12-14). This filter generates a two-dimensional dataset of size 1024x1024, which is subsequently given to an importance function, `etkMagnitudeAF`. This computes the magnitude of each of the vectors in the slice (lines 18-19) and is used as the importance representation provided to the `etkStandardDistributor`, which in turn uses the standard distribution method to compute the distribution pyramid. The number of glyphs to distribute is set to 14000 (lines 23-25). The output of the distributor is then given to a line glyph object, `etkLineGlyph`, which uses the mask data provided by the distributor to determine where glyphs should be drawn. Glyphs are drawn using the data from the slice extractor as input and by binding the x, y, and z components of the data to the three variables of the line. The color variable is unbound, so the glyphs are displayed in the default color (blue) supplied (lines 30-35)

ETK is designed to be used by an experienced visualization programmer, i.e, someone who is familiar with computer programming and the fundamentals of data visualization. In this context the visualization programmer works in conjunction with a researcher to build the required visualization, much in the same way a software developer works with a client to produce a software system. To construct a visualization, the programmer selects the desired processes to use and builds the network by connecting the input of downstream objects to the output of relevant upstream objects. Using the built-in execution mechanism, the network will produce the desired output when the renderer is asked to display the data. This example can be easily extended by adding new data filters and data sinks to the network. One useful extension is to use `etkOutlineFilters` to produce outlines of both the extracted slice and the data volume itself. This would be achieved by creating two instances of the `etkOutlineFilter` class and setting the inputs of these filters to the output of the reader and the slice extractor. To display these outlines an `etkGeometryMapper` is connected to the output of each outline filter. These mappers are then added to the renderer, which will also ask them to display themselves when the display method is invoked. This display method initiates a pull/push cycle through all process objects generating the data needed. The user can interact with the objects by panning and zooming (which is supported by the renderer) the generated display. Although this function is only 40 lines long by using the functionality provided by ETK it can render a dataset with 1 million vectors in an image of dimensions 256x256 while maintaining the ability to see interesting patterns and structures in the field. However although it may be straight-forward for a visualization programmer to create such a program, the raw source code does not provide an easy mechanism for the visualization user to manipulate the parameters of the visualization or otherwise change how the data is processed and displayed.

5.6 Summary

In this chapter I introduced ETK - the Eikon Toolkit, which can be used to render large and multivariate data using geometric coding methods. Although geometric coding is an attractive method for displaying multivariate datasets, it has a number of problems when displaying large amounts of data relating to the amount of space available to display each glyph, the specific geometric properties of each glyph, and the need to display potentially millions of geometric primitives. I described how ETK addresses these problems by selectively displaying glyphs based on the location of features of interest within the data. As a result, global overviews of the data are obtained where important information is visible through the density of glyphs in the display. By zooming areas of interest, intermediate and elementary views are obtained at which point visual variation in individual glyphs can be seen. I described the design of ETK and showed how this design could be extended by adding new abstraction, importance and data glyph objects. The addition of new objects in these cases is a matter of overriding a single method in each class.

Chapter 6

The Visual Programming Environment

The Eikon Toolkit, discussed in the previous chapter, provides a low-level programming interface supporting the implementation of visualizations using the C++ programming language. In this chapter, I introduce the second part of the Eikon system, a visual programming environment that provides a high-level programming interface for the construction of visualization networks based on ETK. The interface uses tools, that are abstractions of individual or groups of processes in ETK, and allows these tools to be visually connected and configured to produce visualization networks. The programming environment allows new tools to be added whenever new processes are added to the underlying ETK or when new abstractions on processes are required. In the remainder of this chapter I describe the goals of the visual programming environment and describe the system architecture and design used to implement it. I also show that the visual programming environment provides both high-level and low-level methods of constructing and configuring visualization networks.

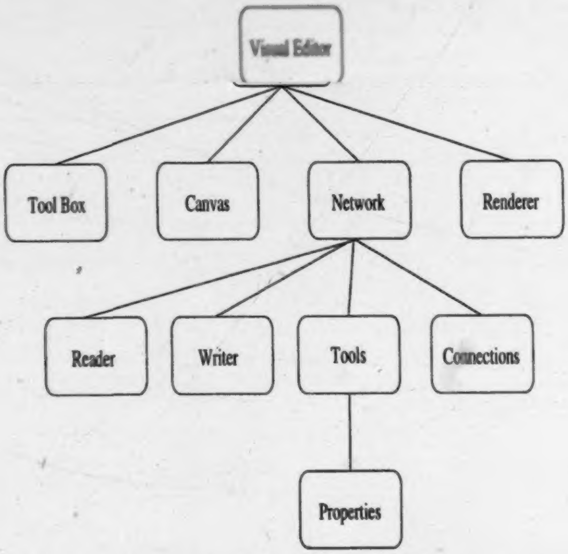
6.1 Introduction

Creating visualization networks directly in C++ provides a high degree of flexibility and power. This type of interface allows programmers to use the complete functionality of the programming language to realize their visualizations. With this degree of flexibility and power, however, comes an increased difficulty in creating the visualizations, understanding the data flow and processes within the network, and exploring the range of parameters offered by the network. The low-level programming approach lies at one end of a spectrum of approaches that can be used to construct visualizations. At the other lie custom "turnkey" visualization applications specifically tuned to support the visualization of data from a specific domain. For example, the FAST system [banc90] provides support for common visualization algorithms associated with computational fluid dynamics. Such systems are very useful when applied in their respective domain, but are restrictive when data from other domains must be displayed.

Visual programming environments for constructing visualizations have become very popular as an alternative to low-level programming or turnkey applications. This type of visual interface, in which visualization networks are constructed by selecting and connecting various "modules", can be found in most existing commercial products that attempt to combine the power and flexibility of a low-level programming approach with the ease of use of turnkey applications. The visual editor typically excludes access to source code and precludes the use of some of the functionality provided by the low-level programming environment. In addition, the large number of modules available, and the lack of detailed knowledge about the techniques each encompasses, can make the construction of visualization networks difficult.

Visual Programming using ETK

Eikon supports a visual editor for constructing visualization networks. The objective of the editor is to allow users the ability to readily explore the data by manipulating the various parameters of networks which are provided by an expert user or visualization programmer. The editor allows access to every variable of each process in ETK, and allows visualization networks to be exported as C++ code so that they can be further refined. Like other visual editors, it also provides higher-level mechanisms for constructing networks. The editor represents individual or groups of processes from ETK as "tools" and allows networks to be constructed by graphically connecting tools together. In this way, prototype visualization networks can be constructed quickly and the visual depiction of the network presents a method of "visualizing" the visualization network itself. Visualization is an exploratory process, not only in terms of data exploration but in terms of exploring different methods for transforming and



The visual editor system has four agents: (1) a tool box, (2) a canvas (3) a network, and (4) a renderer. Each of these parts is responsible for a subset of the functionality of the editor and each part in turn is defined in terms of a number of lower-level agents that support the functionality of the higher-level agent.

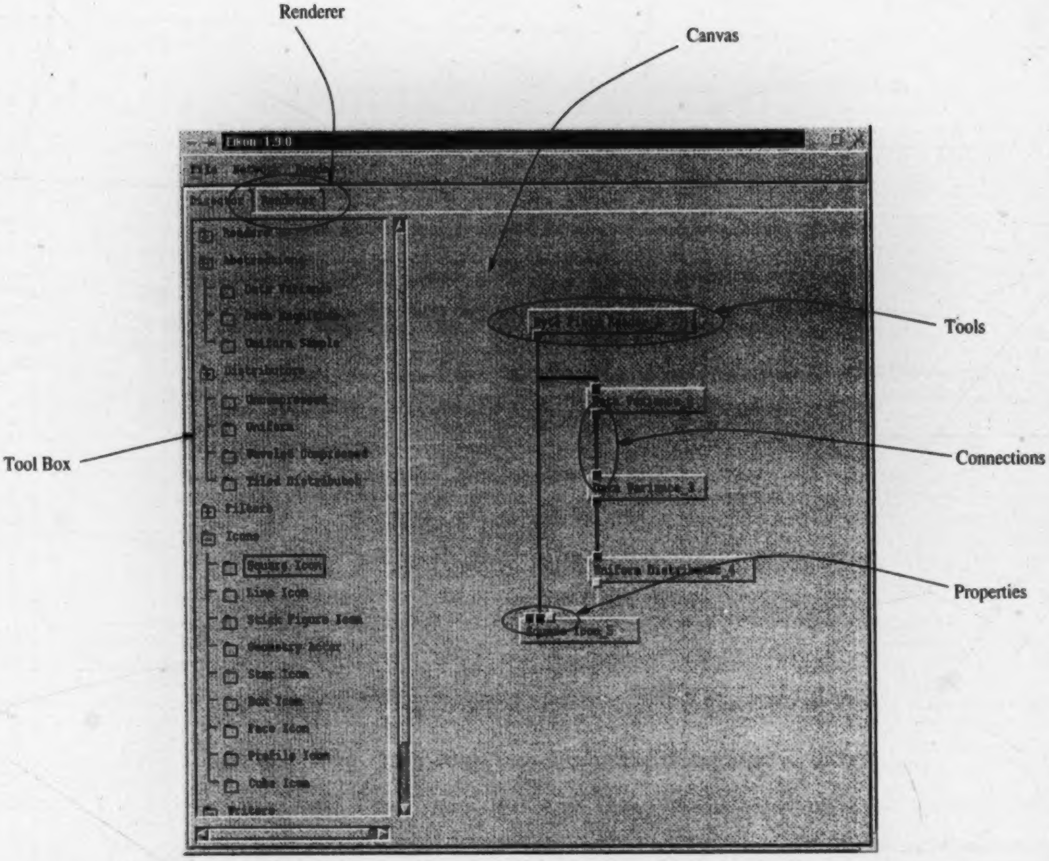
Figure 6.1: The system architecture of the visual editor.

representing data.

6.2 System Overview

An overview of the visual editor is shown in Figures 6.1 and 6.2. The system is based on a version of "presentation-abstraction-control" or "PAC" architecture described in [busc99] and is defined in terms of a hierarchy of self-contained cooperating objects, or agents.

The advantage of organizing the system as a hierarchy of cooperating agents is that it allows the system to evolve over time. Within Eikon, and similar visualization systems, this is very important; the underlying toolkit will change with the addition of new data filters, abstraction functions and glyphs, and each of these new processes must be easily integrated into the visual editor. By organizing the editor into a hierarchy of self-contained agents this process is easier. For example, from a visualization network viewpoint, the top level editor is only concerned with network construction



Each agent defined within the editor has a corresponding user interface component. This figure shows the tool box, the canvas, tools, connections, and properties. The renderer is invoked by clicking on the "Renderer" tab and network reading, writing, and code generation is accessed from the main menu.

Figure 6.2: The visual editor layout.

and execution. Networks themselves are concerned only about the tools that define them and their connections; tools are only concerned with the processes they represent and their properties. Adding new tools requires a new class that describes the relevant properties of the processes that it represents. This does not require any change to the network or to the editor only the addition of tool creation methods to the tool box agent. In addition the PAC architecture supports multiple levels of presentation of information associated with the visualization network. The network is displayed as a series of connected tools - where tools and their properties are displayed as small icons and where the parameters of individual tools can be examined and modified through associated configuration dialogs. The ability to represent visualization networks as collections of tools, and the ability to examine and modify all parameters of each tool, supports both high-level and low-level programming approaches.

The top level PAC agent is the visual editor application itself. All other agents depend on this application agent to operate. The top-level agent maintains the static user interface to the system and a number of other common variables that describe the current state of the editor. The system contains four intermediate agents which correspond to the four major vertical divisions of functionality within the editor.

1. The Tool Box

   The tool box represents the collection of tools available for the construction of visualization networks. The tool box has an associated user interface component (a list of tools) from which the user can select tools to place in the network. It organizes the tools into collections corresponding to *Data Sources*, *Data Filters*, *Abstraction Functions*, *Distributors*, *Glyphs*, and *Data Writers*. Tools are constructed by the tool box by using an associated tool factory object.

2. The Network

   The network represents the visualization network being constructed by the user. It contains a collection of tools and their connections, that in turn define the processing the network will carry out. The network agent also uses lower-level agents to allow networks to be read from and written to secondary storage.

3. The Canvas

The canvas agent provides a graphical representation of the network under construction. Each of the tools and connections defined in the network has its graphical representation drawn on the canvas. Networks are created by selecting tools from the tool box and placing them on the canvas. Tools are connected by drawing links between them, and can be modified by clicking on their graphical representation. Any tool or connection made on the canvas is also added to the network.

4. The Renderer

   The renderer allows the output of the created network to be displayed and interacted with. The render sends a request for data to the network defined in the network agent and ultimately results in glyphs being drawn that represent the data.

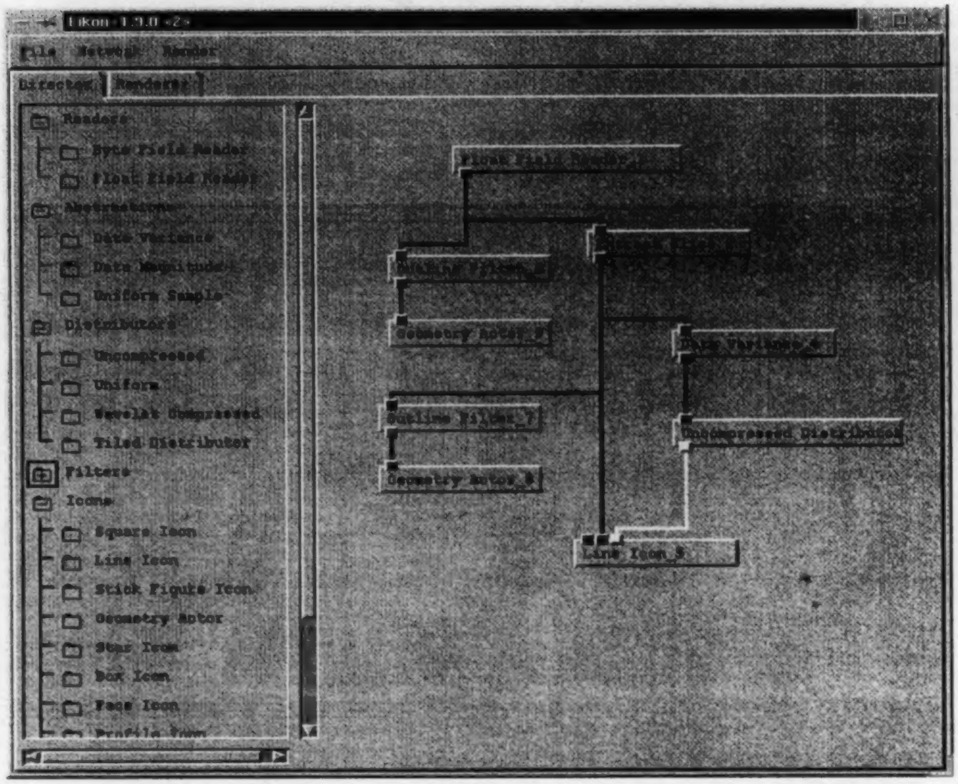
6.3 Visualization Networks

An example of a visualization network created with Eikon is shown in Figure 6.3. This network is similar to the example presented in Chapter 5. It reads a float field from which a two-dimensional slice is extracted. A variance importance function is applied to this slice, which is then used by a standard distributor to distribute line glyphs over the slice. The example also uses an outline filter to render the outline of the input volume and extracted slice to provide context for the displayed data.

6.3.1 Creating Networks

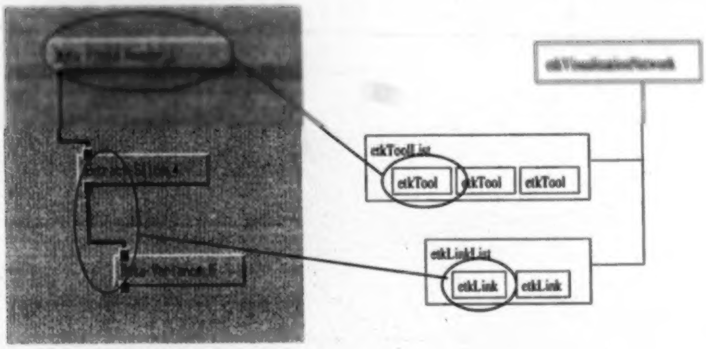
Networks are created by adding tools to the network agent and defining connections or links between them. The network agent represents the network as lists of tools and their links as shown in Figure 6.4.

Within the editor networks are constructed by selecting the appropriate tools from the tool box and placing them on the canvas. Placing a tool on the canvas adds it to the list of tools in the network, creates default instances of the ETK processes it represents, and places a push button representation on the canvas. The push button depicts the process or processes from ETK and contains colored tabs representing



A simple network that extracts a slice from a data volume and renders it using an importance function based on data variance. Outlines of the data volume and extracted slice are also rendered.

Figure 6.3: An example network in the visual editor.



A network is defined by a collection of tools and the connections between them. The network is represented visually as a collection of push buttons with colored lines defining the data-flow connections between the processes. The color of the lines represents the type of data moving between processes.

Figure 6.4: Visualization network representation in the visual editor.

the inputs and outputs of the tool. The tabs are color-coded to indicate the different data types that they expect. Connections between tools are created by clicking on a source tool output tab and drawing a line to the input tab of a destination tool. If the input tab doesn't already have an associated connection, and the input and output types of the tools match, then a connection is formed between the tools. The connection is displayed as a colored line on the canvas that indicates the type of data that flows through the connection. The `SetInput()` and `GetOutput()` methods of the associated ETK processes are called to form the actual connections between the processes and a link object is also added to the list of links maintained by the network agent. Constructing networks using this type of visual programming clearly shows the data-flow paths through the network and the processing that is applied to them. In addition, the color-coding used by the data flow paths indicates the type of data that flows through each connection. This provides a high-level abstraction on the processing and rendering that is carried out which in turn helps explain the nature of the images of the data created.

6.3.2 Modification and Rendering

Clicking on the push button associated with each tool activates a configuration dialog that allows the internal properties of the tool to be modified. The configuration dialog for each tool is constructed from the list of properties defined for the tool. Each property has an associated interface component that allows its value to be displayed and manipulated. Modifying the value changes the tool property and any associated process variables. The ability to examine and modify the individual parameters of each tool provides a lower-level more detailed view of the visualization process and enables the more subtle visual variation in the image created to be understood and manipulated.

Once a network has been created and configured, an image of the data can be generated by selecting the "Render" tab from the user interface. This causes a request for data to be sent to all data sinks in the network and generates an image of the data through the pull/push scheme defined in ETK. New images of the data can be generated by changing the parameters associated with each of the tools in the network, or by adding or deleting tools in the network. Modification of tool properties causes the modification time of the underlying process objects to be updated, ensuring that new output is generated the next time the data is displayed. The renderer allows the user to rotate, pan, and zoom the data displayed. Interaction is also supported by using individual tool configuration dialogs from the editor. When values of tools are changed using the editor, the network re-executes the affected tools producing an updated image in the renderer. This allows the user to explore different settings for different parameters in different tools.

6.3.3 Reading, Writing, and Code Generation

Visualization networks can also be saved to disk and loaded from disk by using the file menu from the main menu within the visual editor. Reading and writing networks is supported by the visualization network reader and network writer. The writer writes the network to disk by traversing the list of tools and links defined in the network agent. Each tool is written to disk followed by the list of properties it contains and their current values. After all tools and properties have been written, each link is output by describing its source property, its destination property, and its geometry. Reading a network from disk is simply the inverse process. As tools are encountered, they are added to the network. As links are encountered, links between tools are created and drawn on the canvas.

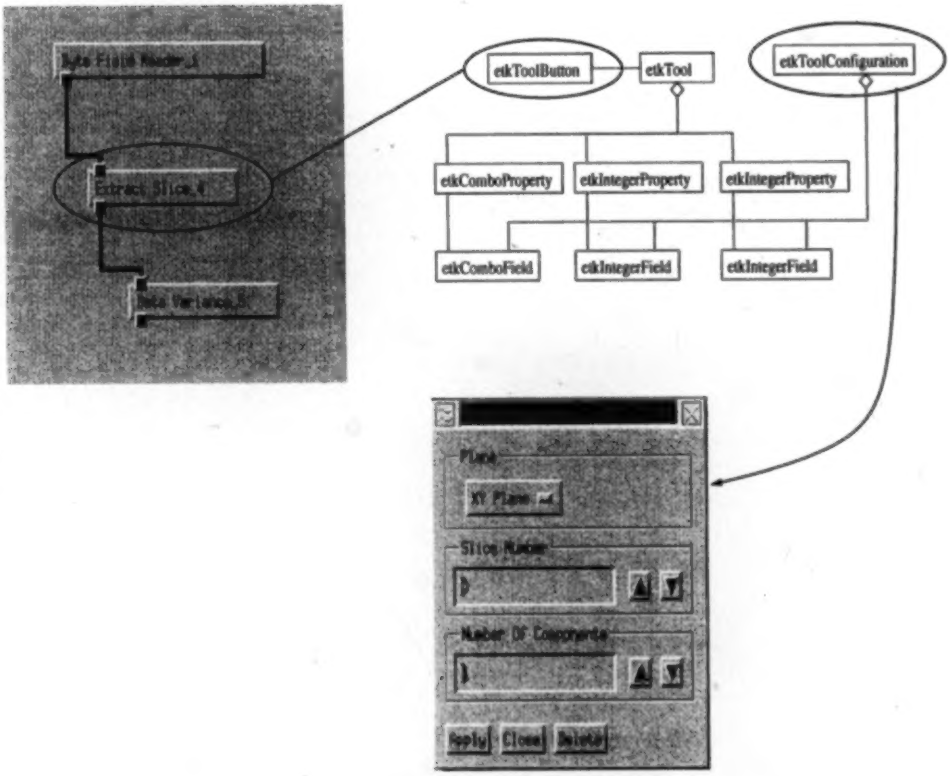
Finally, C++ code can be generated for any network by using the "Generate Code" option from the network menu. Each tool in the editor knows how to generate C++ code using the specified parameters from the tool configuration dialog. The code generation also ensures that the specified links between processes are represented correctly. This option also generates a Makefile, that will compile the generated code with the required ETK libraries, producing a stand-alone executable. When this program is executed, it generates a render window in which the data is displayed and in which the user can rotate, pan, and zoom the data. The current implementation, however, does not allow the user to change internal process parameters.

6.4 Tools, Properties, and Configuration

Tools, properties, and their associated configuration dialogs allow the constructed visualization networks to be configured. Tools represent processes within ETK, properties represent variables associated with each process and configuration dialogs allows values for these variables to be specified. The relationship between tools, properties, and configuration dialogs is shown in Figure 6.5

6.4.1 Tools

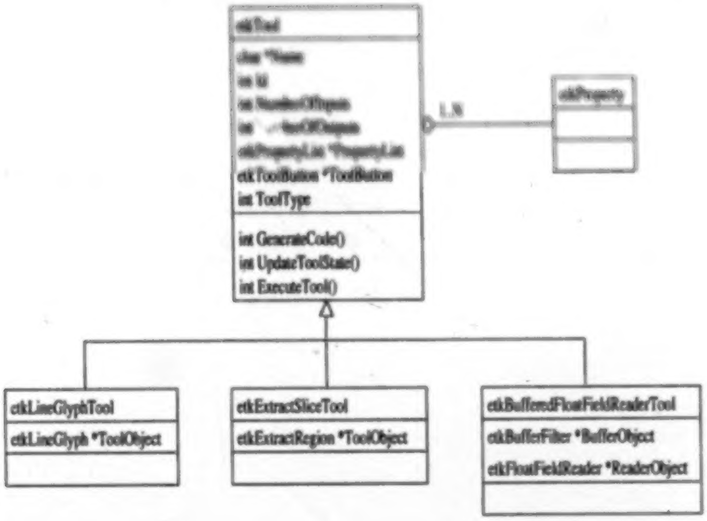
Tools encapsulate one or more processes from the underlying ETK, allow users to modify the parameters associated with these processes, and control the execution of the processes they encapsulate. Each process object in ETK has at least one associated tool in the visual editor. Some processes have several associated tools, and other tools represent two or more underlying processes. For example, the extract region filter has two associated tools: an "extract region tool" and a specialized "extract slice" tool while the "DEX Reader" (Data EXtraction Reader) tool encapsulates several processes in ETK. The "extract region" tool provides complete access to all parameters within the extract region filter, while the "extract slice" tool restricts the parameters of the region to those corresponding to a slice through the data. The "DEX Reader" encapsulates processes that expect data representing pressure and computes velocity and vorticity values from it; producing a dataset that contains all three components. In this way tools provide useful abstractions on individual pro-



Plane: XY Plane
Slice Number: 0
Number Of Components: 1

Tools are defined as a collection of properties. Each property has an associated data field; these are collected together into a tool configuration dialog that allows the properties of the tool to be accessed and modified.

Figure 6.5: Tools and their properties.



Tools are created by sub-classing from the abstract "etkTool" class. The tool class defines a tool as a collection of properties relating to the processes they represent. Concrete versions of the tool class provide pointers to the specific processes they encapsulate and provide the necessary methods for accessing and modifying the tool properties based on their internal variables

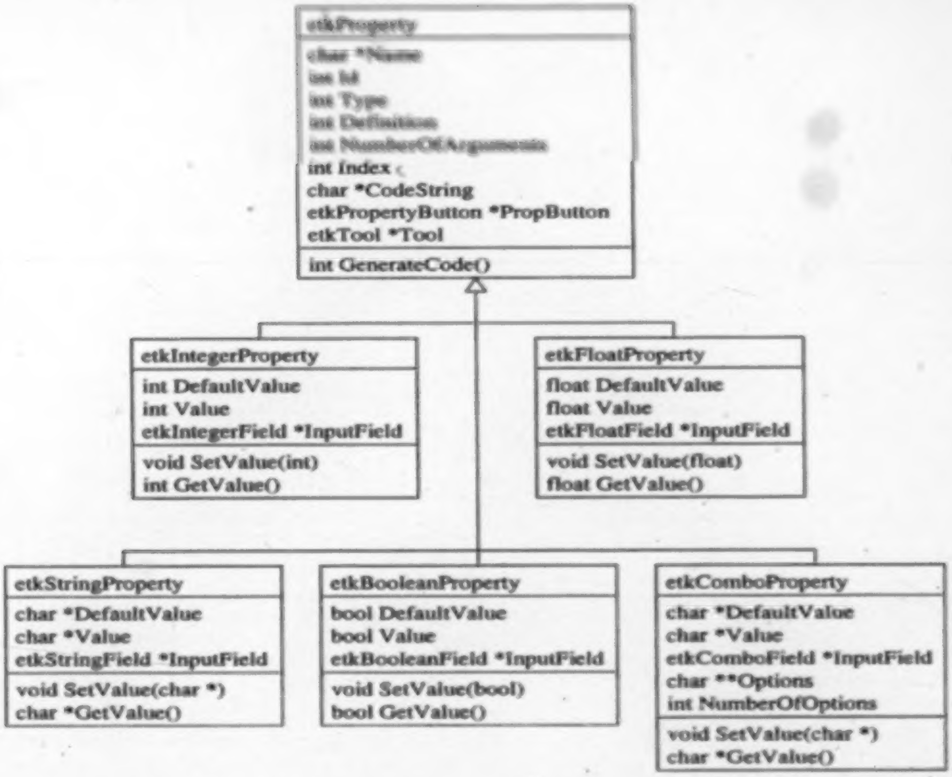
Figure 6.6: The etkTool class hierarchy.

cesses and on groups of processes (or sub-networks), thus simplifying the construction of visualization networks.

Tools are represented by the `etkTool` class as shown in Figure 6.6 and new tools are added to the system by creating concrete sub-classes. Concrete sub-classes must contain pointers to the ETK process objects they encapsulate, define instantiation methods for these process objects, and create an appropriate set of properties to describe the different internal variables of the processes. To enable the use of tools in the visual editor, a description of the tool must be added to the list of tools in the tool box, and a creation method must be added to the tool box tool factory.

6.4.2 Properties

Tools must allow access to and modification of the internal variables associated with the processes they encapsulate. To do this each tool object maintains pointers to the process object it represents and to list of property objects that describe the various characteristics of each processes' internal variables. The property class `etkProperty`,



The visual editor defines five basic properties that correspond to the most common types of data encountered within ETK. If necessary, new properties can be added by creating new concrete sub-classes of the base property class.

Figure 6.7: The etkProperty class hierarchy.

shown in Figure 6.7, represents various characteristics common to all properties, including the property name, type, number of arguments, the index in the argument list where this property occurs, a template for generating C++ code for the property, and a pointer to the tool to which the property belongs. Concrete properties that represent actual data types for properties are sub-classed from the property base class. The editor currently defines five concrete property types: integers, floats, booleans, strings, and combinations, or enumerated types.

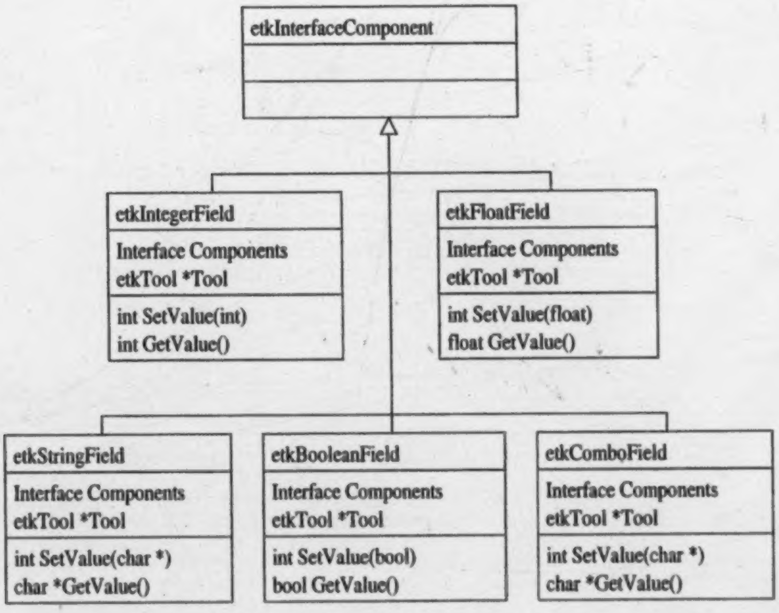
Each property also maintains a definition that describes higher-level semantics associated with it. Properties can be classified as input, output, internal, or hidden. Input properties correspond to member variables that represent the external inputs of the process associated with the property; output properties correspond to the external outputs of the process; internal properties correspond to the internal member variables; and hidden properties are properties used to configure the processes encapsulated by the tool. The definition of the property governs how it is represented in the visual editor. Input and output properties are represented on the canvas as small colored tabs attached to the tool button; internal properties are represented by user interface components called *Data Fields* that allow the values of the properties to be viewed and edited; and hidden properties have no representation, they contain configuration information not accessible to the user, such as the number of options in a combination property for example.

For each tool, the input and output properties maintain the type of data that they accept or that they can generate. These are used to color-code the input and output tabs on the canvas, and to ensure that the output of a specific tool can only be connected to the input of another tool that accepts the corresponding data type. This provides an extra level of type checking and reduces potential errors in the network.

For most tools, the internal properties form a one-to-one correspondence with the member variables of the encapsulated process objects. However, this is not always the case: internal properties can represent a simplification of the variables actually defined in the process objects. A single value provided by an internal property can be expanded into several internal data variables within a process object. For example, the "slice number" internal property of the "extract slice" tool is used to determine values for the x, y, and z variables of the underlying extract region process.

6.4.3 Data Fields

Each internal data property has an associated graphical user interface component, called a data field, that is used for viewing and editing its values. These interface components are not displayed as part of the tool representation by default but are displayed upon request as part of a configuration dialog associated with the tool. Each data field is derived from the base `etkDataField` class, that provides the necessary methods for managing the various user interface components for each field. Specific sub-classes determine the specific user interface elements that will be used to support the display and modification of the field/property values.



Each of the five properties has an associated data field. The data field simply defines the graphical user interface component used to access and modify the value of the property.

Figure 6.8: The data fields defined in the visual editor.

Configuration Dialogs

Access to the internal parameters of a tool is achieved by collecting all the data fields associated with a tool's internal properties into a tool configuration dialog. The tool configuration dialog is activated by clicking on the "tool" push button on the canvas. An example of a tool configuration dialog is shown in Figure 6.5 that shows the dialog for the "extract slice" tool. This dialog contains one combination dialog, for selecting one of three planes to extract the slice from; an integer data field to specify the slice number; and another integer field to specify the number of components, or data variables, to extract from the slice. Through attaching call-back functions to the various components of each data field, the property values they represent can be manipulated. The tool configuration dialog also allows these changes to be applied to the internal variables of the associated process objects (through the "apply" button) or for the tool and its connections to be deleted from the network (through the "delete" button). Modifying the internal variables of a process object results in the modification time of the process object being changed, and will result in the process object re-executing the next time a request for data is made to it.

6.5 Summary

The visual programming environment discussed in this chapter presents a tool for rapidly prototyping potential visualizations. It allows the visualization to be designed at multiple levels of abstraction, including working with default instances of the tools presented in the environment, configuring individual parameters of process objects from the underlying ETK, and working directly with ETK in exported C++. Visualization networks are constructed and depicted graphically, allowing the data flow and processing carried out to be readily seen and manipulated. The tools within the programming environment also allow access to each of the variables within each process, providing a way to readily explore the parameters of constructed visualization networks.

To support flexibility and extensibility, the visual programming environment is designed as a hierarchy of cooperating agents. At the lowest level of the hierarchy are the individual properties that describe the parameters of each process object. At the intermediate levels, properties are grouped into tools that describe individual or collections of processes in ETK, and tools are grouped into visualization networks that perform the necessary data transformation and display. At the top level, the visual editor groups all components together into a visual programming application. Each level in the hierarchy is only loosely coupled to the level above or below. This design allows new properties to be added at the lowest level when new data types need to be represented, and new tools to be added at intermediate levels when new processes are added to ETK. This design therefore provides a flexible and extensible environment supporting the development of visualization networks and the process of visualization itself.

Chapter 7

Case Study - Introduction

In this chapter I introduce the case study conducted to evaluate the usefulness of Eikon as a system for using geometric coding to visually explore and understand large multivariate data. The study was undertaken with researchers in the fluid dynamics research group in the Department of Mathematics at the University of Alberta and consisted of the case study participants, working with the author in the application of Eikon to the visualization and exploration of results generated by a geophysical fluid dynamics (GFD) computer simulation. In the following sections I present the background of the application domain, why this application domain is a good case study candidate, describe the objectives of the GFD research, and the objective of this case study in the context of this thesis. This sets the stage for the next three chapters where I detail how Eikon was used to produce various views of the GFD dataset. These chapters provide motivation for the different views in the context of the case study, describe how the views are created, describe how the various parameters that affect the views can be modified, and provide several examples of the types of images generated. At the end of each chapter salient comments from the researchers involved are discussed and the advantages and disadvantages offered to them by Eikon are presented. In Chapter 11 the results of the case study are collated and conclusions drawn.

7.1 Introduction

One of the goals of the fluid dynamics research group is to increase the understanding of large scale oceanographic and meteorological dynamics. The Navier-Stokes equations, which describe the fluid dynamics of such processes, are very complex and can't be solved analytically or numerically on the scales the fluid dynamics group are interested in. As a result physical and mathematical assumptions are generally used to determine which terms in the equations are important and which terms can be simplified. With such simplifications tractable mathematical models can be constructed that simulate fluid dynamics, generating results closely resembling observed physical behavior. Comparing the results generated by the simulation to actual measurements of the physical behavior, and expected results based on accepted theory allows the verisimilitude of the model to be tested. Results closely matching those of physical behavior provide evidence that the assumptions regarding important terms for predicting fluid behavior are valid. If the results do not closely match those of physical behavior it indicates that the assumptions made may be false. In either case the ability to compare, contrast, and otherwise analyze the results can provide insight into the accuracy of the current models and highlight ways in which the accuracy can be improved.

7.2 Density-driven Benthic flows

Of particular interest to the fluid dynamics group is modeling the evolution of density-driven benthic currents in the world's oceans. These currents are topographically steered currents of dense water normally associated with sloping continental shelves, mid-ocean ridges, and canyons. Examples of benthic flows include the northward movement of Antarctic Bottom Water from southern polar regions [hain98], the bottom trapped currents in the Strait of Georgia between Vancouver Island and mainland British Columbia [stac87], and the Denmark Straight Overflow located between Greenland and Iceland [bruc95]. Understanding the evolution of benthic flows is important for many reasons. Benthic flows are responsible for transporting heat, oxygen, salt, sediment, and nutrients over great distances; they can also affect weather patterns and longer term climate variability. Benthic flows also play an important role in thermohaline circulation, which is a circulation that acts like a global heat pump moving warm salty water from the equator to the poles where it cools, sinks to the ocean floor, and moves towards the equator where the cycle begins again.

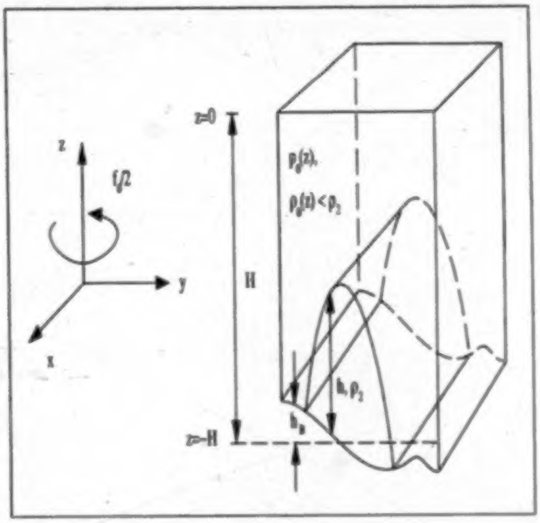
7.3 Model Overview

Intuitively the behavior of such topographically steered flows can be understood by considering the behavior of fluid flowing along a slope. In the simplest case the fluid tends to flow down the slope under the force of gravity, losing its potential energy in the process. In the Earth's ocean this behavior is more complex as the Earth's rotation introduces the "Coriolis" force that tends to deflect the fluid flow to the right (in the northern hemisphere) or to the left (in the southern hemisphere). At some position, the deflection introduced by the Coriolis force will cause the body of fluid to stop flowing down the slope and flow along it instead. These flows are referred to as geostrophic flows indicating the balance between gravity pushing the flow down the slope and the Coriolis force pushing it along the slope. In the Earth's ocean such flows are more or less permanent, however, they can undergo periods of instability that affect their behavior. Instabilities are the result of small disturbances in the flow that grow over time, and can result in the development of features such as mesoscale waves, meanders, and eddies, often causing the currents to spread out or move offshore. The fluid dynamics group are interested in predicting and characterizing the evolution of these instabilities. In particular they are interested in baroclinic instabilities, and have developed a mathematical model that attempts to predict their evolution. If at any time a fluid parcel moves further down the slope a portion of the gravitational potential energy that it contains is released making it available to form baroclinic instabilities. These instabilities can evolve into a number of features that can dramatically affect the characteristics of the flow in general

A simple schematic[1] of the mathematical model is presented in Figure 7.1. The model represents the ocean as a two-layered fluid with the bottom layer being more dense and less thick (in terms of height) than the upper layer. Several two-layered fluid models exist that use several assumptions to simplify the governing Navier-Stokes equations, however these assumptions are based on little variation in the interface between the two fluid layers with respect to the total thickness of each layer. In the case of benthic flows the variation in interface is large with respect to the bottom

---

[1] We present a much simplified outline of this model, the precise details are outside the scope of this thesis and can be found in [resz97, resz99]



The ocean model used to produce the GFD data. This simulation describes the circulation within the ocean over time. (Image provided by the fluid dynamics research group)

Figure 7.1: The ocean model.

fluid layer, i.e, it can be as much as the thickness of the bottom layer itself, but it is small with respect to the overall thickness of the top layer.

To reduce the inaccuracies of using existing models [haid91], the fluid dynamics group have developed a new model that uses a system of four coupled (partial differential) equations to model both fluid layers. As the variations in the interface between the two fluids are small compared to the top-layer thickness, continuously-stratified quasigeostrophic equations are used to represent this layer while the lower layer is modeled using the Shallow Water equations (see [resz99] for precise details.) The equations of both layers are based on assumptions that the fluid is incompressible, that pressure is a function of the height of the water above above and the Coriolis force is constant. In addition the Shallow Water equations assume that the fluid is homogeneous and horizontal scales are much larger than vertical scales while the top layer equations assume that density can be taken as constant except where it multiplies gravity or vertical velocity (a Boussinesq fluid), the fluid is roughly in geostrophic balance, vertical velocities are small and variations in bottom topography are small compared to the total fluid depth.

7.3.1 Generated Data

A computational simulation using this model was developed by one of the case study participants.[2] The result of this simulation is a dataset consisting of 200 time slices depicting the evolution of instabilities over time as predicted by the mathematical model. The data used in this case study consist of a collection of 1024×1024 grids containing pressure and height values sampled from the generated data. This type of data corresponds to a structured point dataset and is common in computational fluid dynamics. Eikon has been specifically designed to work with such datasets.

7.3.2 Model Verification

To determine if the understanding of the instabilities and their evolution is correct the output of the model must be compared and contrasted with observed real world phenomena; for example, measurements from the Strait of Georgia [stac87] or the Denmark Strait Overflow [bruc95], and with expected behavior based on accepted theory. Visualization plays a crucial role in this context by providing researchers with the ability to observe structures and patterns in computed results and to compare these with real world phenomena. Visualization also aids in correlating the results with expectations and can help ensure that the numerical scheme is working properly. If unphysical results are observed this may indicate a problem with the code. These simulations produce large datasets, both in time and in space, therefore any software used must be capable of processing and displaying large amounts of data at once.

7.4 Case Study Objective

The objective of this case study is to show that Eikon can be usefully applied to the visual exploration and analysis of large multivariate datasets by using abstraction, importance and distribution of glyphs to extend the use of geometric coding to large datasets similar to the GFD dataset discussed here. This objective is met by describing the various needs of the case study participants and indicating how these are satisfied using the functionality of Eikon. In the next few chapters I discuss how

---

[2] It was developed and implemented by Matt Reszka, a PhD Candidate in the Department of Mathematics, as part of his doctoral research.

Eikon was used to analyze and explore the GFD data at a number of different scales. I describe how images of the data focusing on various important characteristics of it where generated and how these views support global, intermediate, and elementary readings. In addition I describe how the glyph distribution process in Eikon can produce images that support interactive exploration of the data (using panning and zooming) and how the visual programming interface facilitates analysis and exploration of the data by allowing different visualizations to be constructed and the various parameters affecting the image to be quickly and easily modified.

Chapter 8

Case Study - Elementary Views

One characteristics of an effective image for visualization is that it allows information concerning the data to be read at elementary, intermediate, and global levels. Elementary, intermediate, and global views represent regions on a continuous scale relating to the amount of information visible in an image. In this case study I consider views that contain a small fraction of the data points (typically less than 1%) and that display all the variables in detail to be elementary views. I consider views that represent more data points, up to and including all of them, but only display a subset of the data variables (reduced elementary detail), with potentially less accuracy than elementary views, to be intermediate views. Finally I consider views that represent all the data points but only display one or two of their characteristics to be global overviews.

With respect to the case study, elementary views of the GFD dataset allow precise details of the dataset to be illustrated. This is important as these images can communicate details about the dataset to others and allow the researchers to compare and contrast the data values at different data locations against those they expected to be there. This can increase confidence in the underlying mathematical model or indicate problems that need to be addressed. Geometric coding is effective at constructing elementary views of multivariate data as multiple variables can be represented in a single image. Other techniques, such as color coding, require several different images of the data to be compared to recover relationships between data variables. Existing systems are also readily capable of generating various elementary views of datasets using geometric coding methods. The objective of this chapter is to show that Eikon can also produce these views. In later chapters I will introduce new views which are typically not possible using a single existing visualization system but can be generated within Eikon.

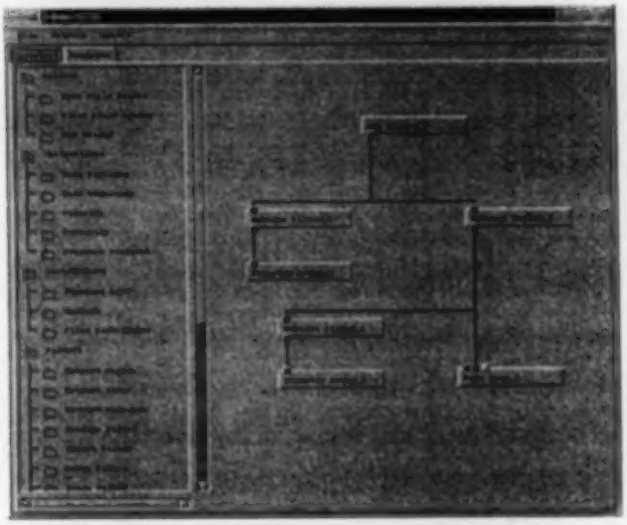
8.1 Elementary Views in Eikon

Figure 8.1 shows a typical visualization network used to produce elementary views in Eikon. The network uses a face glyph to display a small section of the dataset. A DEX reader is used to read the GFD data and produce a dataset in Eikon consisting of height, pressure, velocity, and vorticity components. The extract region tool extracts a 40×40 region from the center of the dataset and this data is provided directly to a face glyph tool. As the face glyph tool does not have a corresponding mask for this data, a face glyph is displayed at every data point in the extracted region. The position of each face is determined from the origin of the input dataset, the position of the extracted region, and the inter-data point spacing. An outline of the entire dataset and the extracted region is determined by two outline filters and the outlines are drawn using corresponding geometry mappers. The outlines provide a context for the data and extracted slice.

The data variable to glyph parameter binding (data-glyph binding) is specified using the tool configuration dialog shown in Figure 8.2 and described in Table 8.1. The face glyph can represented many data variables so several of the input data variables are represented by more than one parameter of the face glyph.

Example output of this network is shown in Figure 8.3 and 8.4. Figure 8.3 shows a zoomed view at the center of the extracted region, while Figure 8.4 displays the entire extracted region.

In Figure 8.3 only 256 glyphs (or 0.024% of the data points) are displayed, each glyph occupying a region approximately 45 pixels by 45 pixels. As a result a good range of geometric variation is visible allowing data variation to be seen through variation in the position of pupils, the size of eyes and nose, the size and curvature of the mouth etc. It has been argued however that face glyphs are not appropriate for visualizing variation in individual data variables [wilk00] and are more suited to visualizing multiple variables (data points) as a whole. The reasoning in this argument

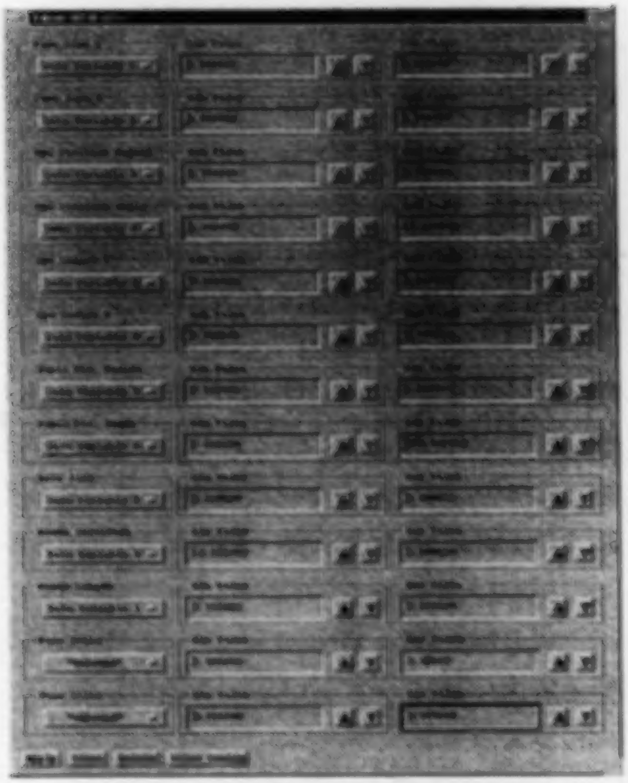


This example network reads in the GFD dataset and extracts a small region near the center. This data is displayed using a face glyph to represent each of the extracted data points. An outline is also drawn around the entire extent of the input dataset and the extracted region.

Figure 8.1: Basic visualization network.

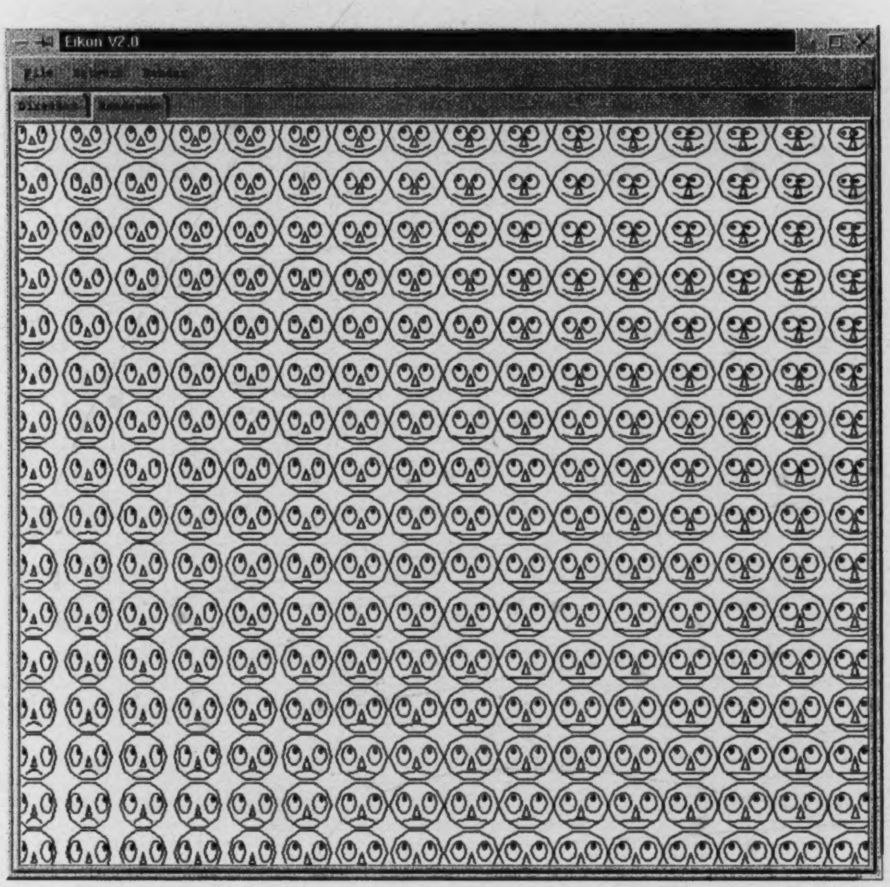
Glyph Parameter	Minimum Value	Maximum Value	Data Variable
Face X Radius	3.0	4.0	Height
Face Y Radius	3.0	4.0	X Velocity
Eye Position r	0.0	2.4	X Velocity
Eye Position θ	0.0	50.0	Y Velocity
Eye X Radius	0.3	2.0	Pressure
Eye Y Radius	0.3	2.0	Pressure
Pupil Position r	0.0	1.0	X Velocity
Pupil Position θ	0.0	360.0	Y Velocity
Nose Size	1.0	3.0	Pressure
Mouth Curvature	-2.0	3.0	Vorticity
Mouth Length	0.1	0.3	Height
Face Scale	0.0	0.3	Unbound
Face Color	0.0	0.0	Unbound

Table 8.1: The bindings for the face glyph (elementary view).



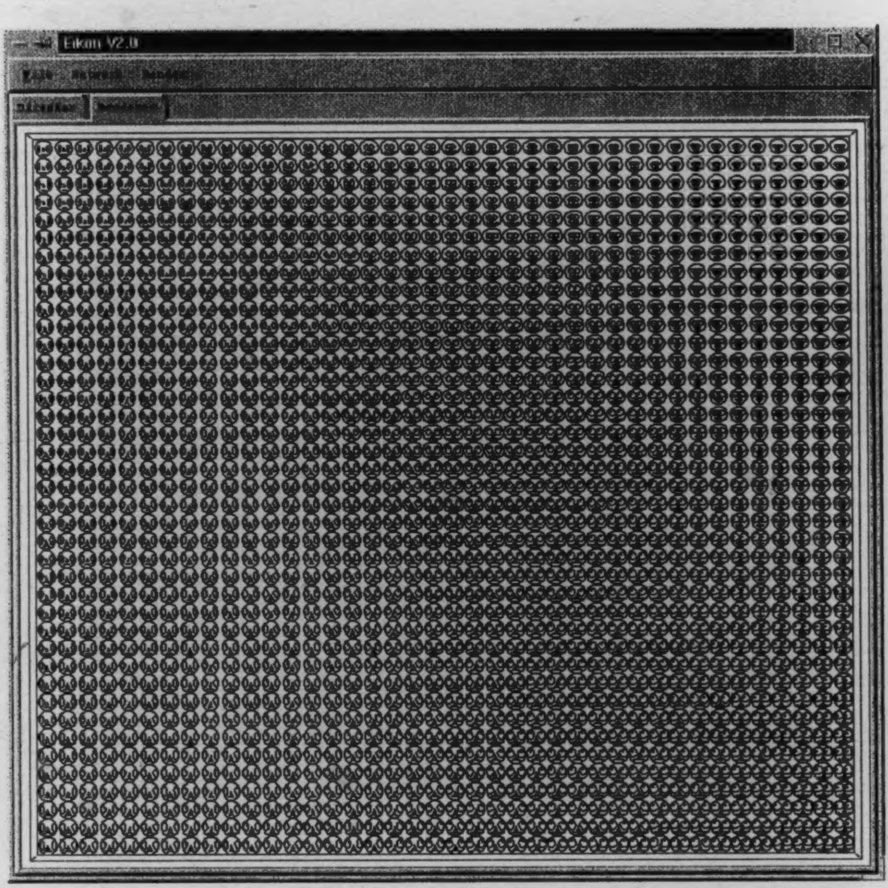
The face configuration dialog used to define the binding between data variables in the GFD dataset and parameters of the face glyph. The binding between data variables and glyph parameters is specified through the option menus on the right. Minimum and maximum values for each parameter are specified through the two text fields. Each configuration tool also contains a sub-dialog for determining color sequences for data display. In this example color is unused (unbound.)

Figure 8.2: An example of a glyph configuration dialog.



The output of the extracted data represented using the face glyph. Only a small portion of the data near the center of the extracted region is shown. At this resolution variation in individual data variables is clearly visible.

Figure 8.3: Zoomed elementary view with the face glyph.



The output of the extracted data represented using the face glyph. Individual variation in facial properties is reduced, however groups of faces varying in gross facial features are evident.

Figure 8.4: Un-zoomed views using the face glyph.

is that the human visual system is "geared" towards processing facial expressions as a whole. Looking at the data points in Figure 8.3 it can be clearly seen that the faces in the bottom of the image have a somewhat sad disposition and those in the top right have a happier expression. It also appears that all faces are looking towards something off the top middle of the image. One problem with face glyphs is that although these variations are apparent the exact semantics of the variation are often hard to determine. This may be due to the non-intuitive link between facial variation and the underlying physical property or even that facial expressions are more ordinarily used to express emotions.

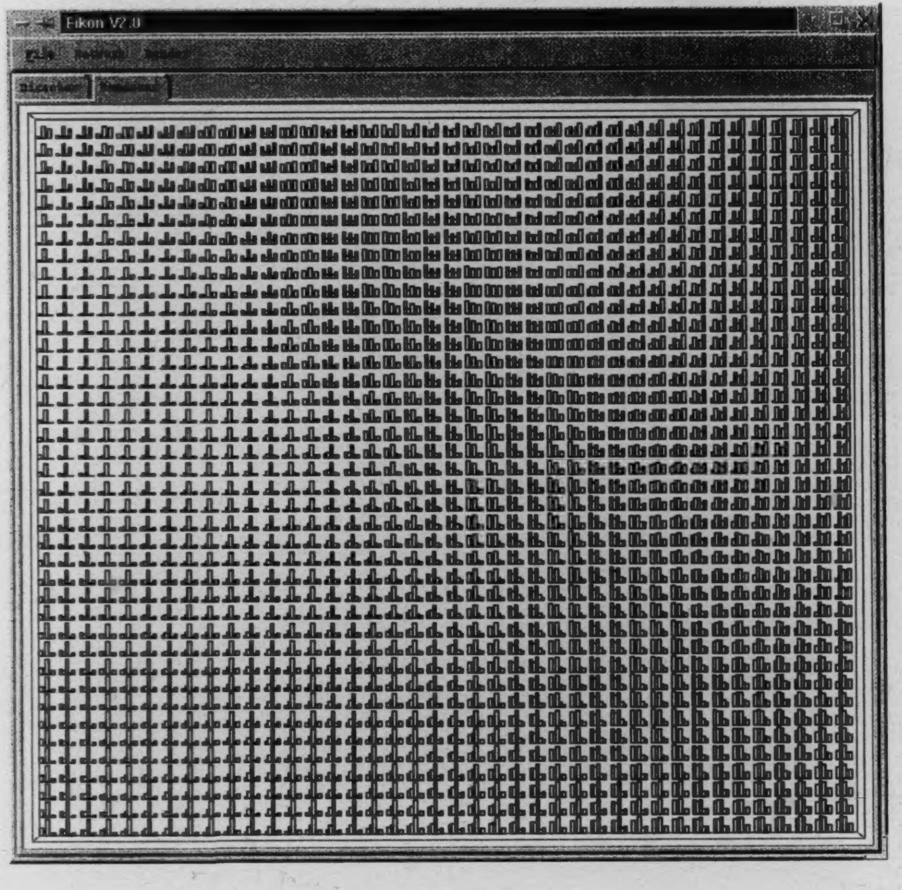
In Figure 8.4 more data points are displayed and although each individual glyph is still visible it is much more difficult to obtain detailed information concerning each data variable and although it is possible to determine groupings of faces based on gross differences in facial properties the amount of visible variation in each glyph is largely reduced making distinction between different facial expressions more difficult. It is however possible to determine groupings of faces based on the variation in texture produced by tiling faces over the image. Although this corresponds to a small scale intermediate view the precise meaning of the texture variation is unclear.

Part of the power of visualization stems from the ability easily to render the data in multiple formats. One way this is supported in Eikon is by allowing the glyph used to display the data and the data-glyph binding to be readily changed. Changing the glyph is a simple matter of replacing the current glyph tool with a new one while changing the data-glyph binding is done through the associated glyph configuration dialog.

Figure 8.5 shows the effects of changing the face glyph tool to a profile glyph tool using the binding described in Table 8.2. This binding maps the data variable to the heights of the bars in each profile. The maximum area of each profile is limited to 0.9×0.9, allowing small gaps between each glyph displayed. The effects of this change are shown in Figure 8.5. In this image each of the 1,600 profile glyphs are still visible, however, unlike the face glyph reasonably precise details of each data variable can still be determined. The magnitude of data variables can be compared with each other within a single data point for example. Unlike the face glyph however it is more difficult to visualize data points as a whole using the profile glyph.

Glyph Parameter	Minimum Value	Maximum Value	Data Variable
Bar 1	0.0	0.9	Height
Bar 2	0.0	0.9	Pressure
Bar 3	0.0	0.9	X Velocity
Bar 4	0.0	0.9	Y Velocity
Bar 5	0.0	0.9	Vorticity
Profile Size	0.0	1.8	Unbound
Color	0.0	0.0	Unbound

Table 8.2: The bindings assigned to the profile glyph(elementary view).



The output of the extracted data represented using the profile glyph. Unlike the face glyph, the simpler geometry of the profile more clearly indicates elementary data variation.

Figure 8.5: Elementary view using the profile glyph.

Glyph Parameter	Scale	Data Variable
X Coordinate of End Point	0.02	X Velocity
Y Coordinate of End Point	0.02	Y Velocity
Z Coordinate of End Point	1.0	Unbound

Glyph Parameter	Minimum Value	Maximum Value	Data Variable
Size of Bulb	0.08	0.15	Pressure
Color	0.0	0.0	Unbound

Table 8.3: The binding assigned to the line glyph (elementary view).

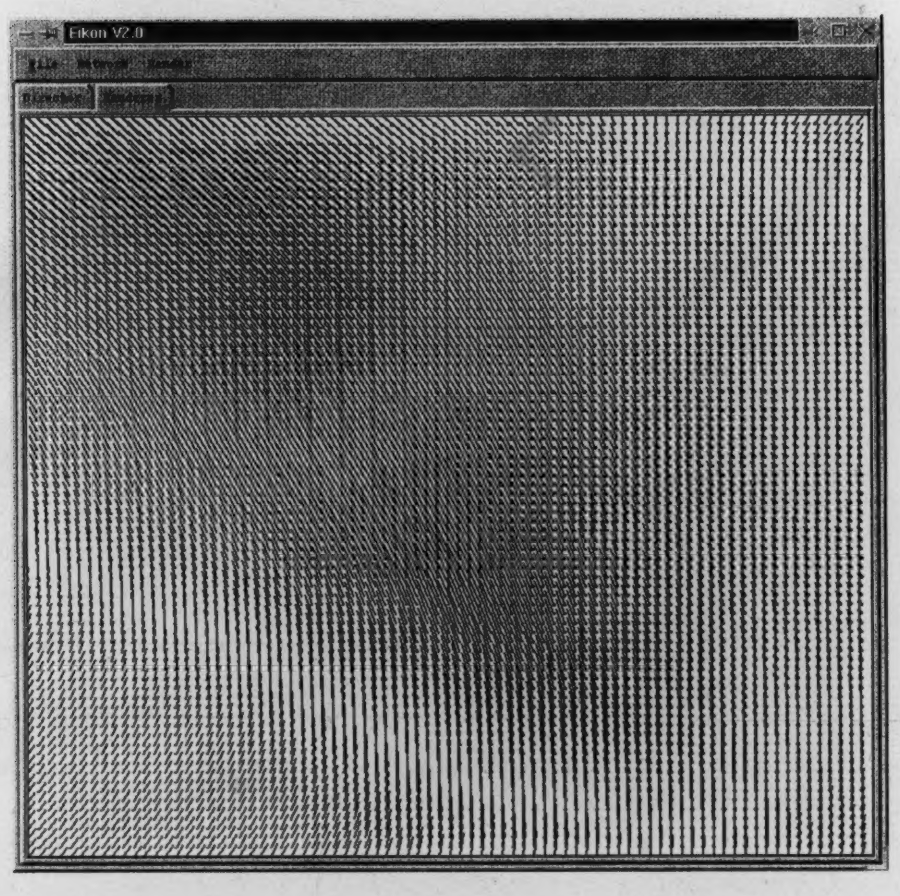
Glyph Parameter	Minimum Value	Maximum Value	Data Variable
Limb 1 Orientation	0.0	180.0	Vorticity
Limb 2 Orientation	0.0	45.0	Height
Limb 3 Orientation	0.0	45.0	Pressure
Limb 4 Orientation	0.0	45.0	X Velocity
Limb 5 Orientation	0.0	45.0	Y Velocity
Limb Length	0.0	2.0	Unbound
Color	0.0	0.0	Unbound

Table 8.4: The binding assigned to the stick figure glyph (elementary view).

8.2 Local Geometry and Texture

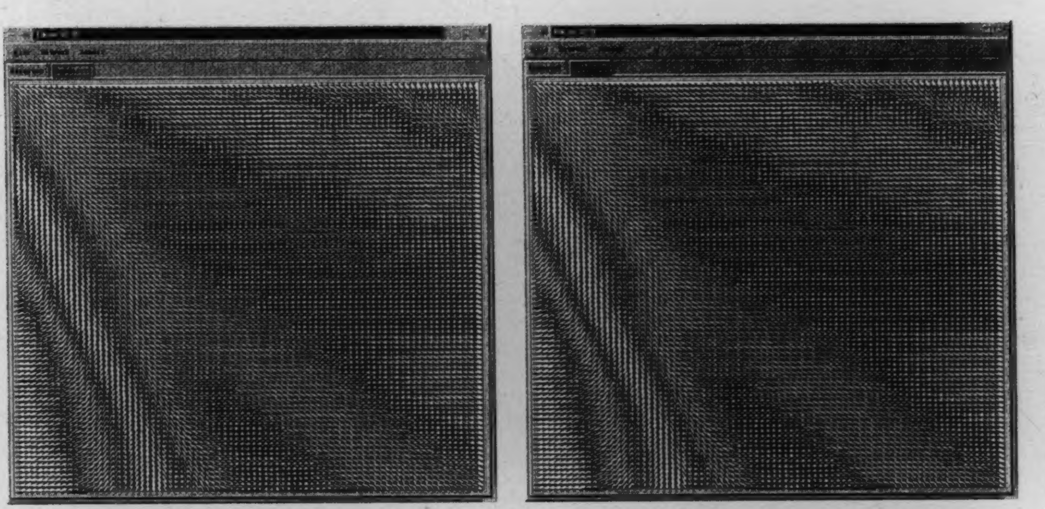
In Figure 8.4 and 8.5 it is evident that at least three distinct groups of data points exist through groups of glyphs that share common features: those in the left of the image, those in the center and those in the right of the image. It is difficult, however, to determine the location of the border between these regions and it is also difficult to determine what is the cause of these groupings. Again this problem can be addressed by changing the glyph, and the data-glyph binding used to display the data. Figure 8.6 shows the results of displaying an extract from the data using a line glyph and Figure 8.7 shows several examples of using a stick figure glyph with different data-glyph bindings. In these images a region 80×80 is extracted from the dataset, four times larger than the previous region, and the line glyph uses the bindings described in Table 8.3 while the stick figure uses binding similar to Table 8.4

Figures 8.6 and 8.7 show the dramatic affect that changing the glyph and its binding can have. Both images contain a visible texture that is indicative of features in the data. Variations in the flow direction and magnitude can be seen in Figure 8.6

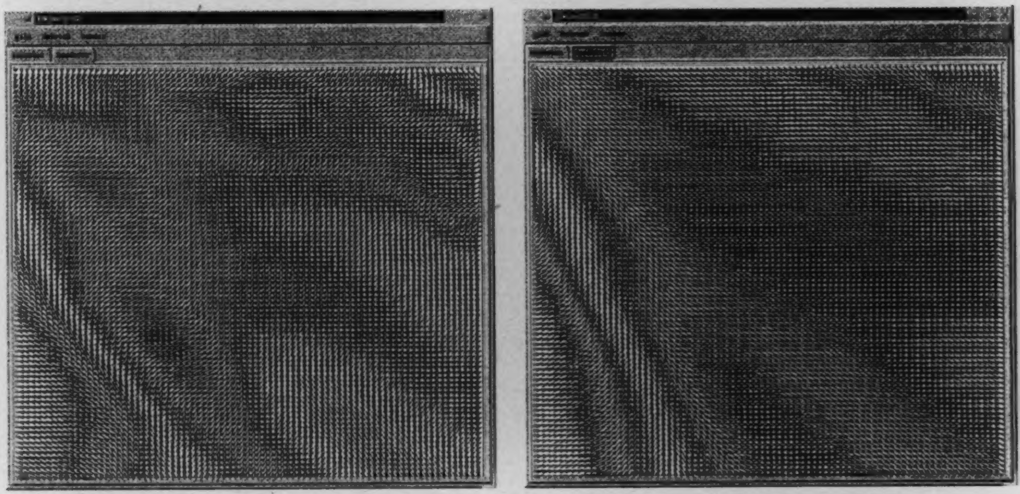


An 80x80 region extracted from position (512,591) in the original dataset. A line glyph is used to display the direction and magnitude of the flow while the size of the bulb represents pressure. This binding introduces a texture reflecting variations in flow direction and magnitude as well as pressure.

Figure 8.6: A larger scale elementary view using the line glyph.



(a) Height (b) Pressure



(c) Vorticity (d) Y-Velocity Magnitude

The stick figure glyph is used to display the data where the main body is bound to (a) Height, (b) Pressure, (c) Vorticity and (d) Y-Velocity components in the GFD data. The displayed region corresponds to an 80x80 area extracted from position (542,512) in the original dataset. In each image a texture is evident reflecting variation in the parameter bound to the stick main body orientation.

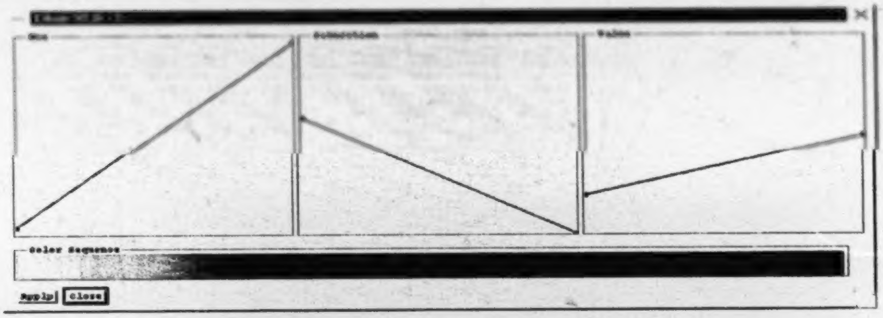
Figure 8.7: Four elementary views using different bindings for the stick figure glyph.

and a resultant texture is also visible. Figure 8.7 shows how several different textures can be generated by simply changing the data-glyph binding used for the stick figure. In this image the texture regions are very apparent but precise details about data variables are masked. Although each of the limbs of the stick figure contribute to the texture generated, a large part of this texture is due to the orientation of the stick figure main body. Figure 8.7(a) uses height to control the main body orientation, Figure 8.7(b) uses pressure, Figure 8.7(c) uses vorticity, and Figure 8.7(d) uses y-velocity magnitude. As a result the texture in each of these images largely reflects variation in these data variables.

8.3 Color

Color is a useful visual variable when displaying large amounts of data and has been used extensively by the participants of this case study to look at variations in scalar quantities from the GFD datasets. Images were generated by specifically designing a color sequence and using this to depict individual variables from the data. Geometric coding can also take advantage of color by using the color of each glyph to represent data variables. In Eikon each glyph tool configuration dialog has a color configuration sub-dialog that allows simple color sequences to be constructed. Color sequences are constructed by graphically creating piecewise linear transfer functions for hue, saturation, and value. Figure 8.8 shows an example of a color sequence, created in Eikon, that varies in hue, saturation and value.

A color sequence is used by binding a data variable to the color parameter associated with each glyph. The bound data variable is used as an index into the sequence to determine the color of individual glyphs. Figure 8.9 shows four examples of using color in conjunction with the line and stick figure glyph to display the data. Each image uses the binding described in Table 8.3 for the line glyph and Table 8.4, for the stick figure glyph. In Figure 8.9(a) the line color is bound to x-velocity magnitude, and in Figure 8.7(b) the stick figure color is bound to vorticity. As a result in both these images color reinforces the texture displayed by the geometric variation of the glyphs. Figure 8.9(c) and 8.9(d) both use the stick figure glyph with color bound to height and pressure respectively. In these cases color depicts data other than that re-



In the color sequence editor the user plots transfer functions for each of the hue, saturation, and value components to define elements in the sequence.

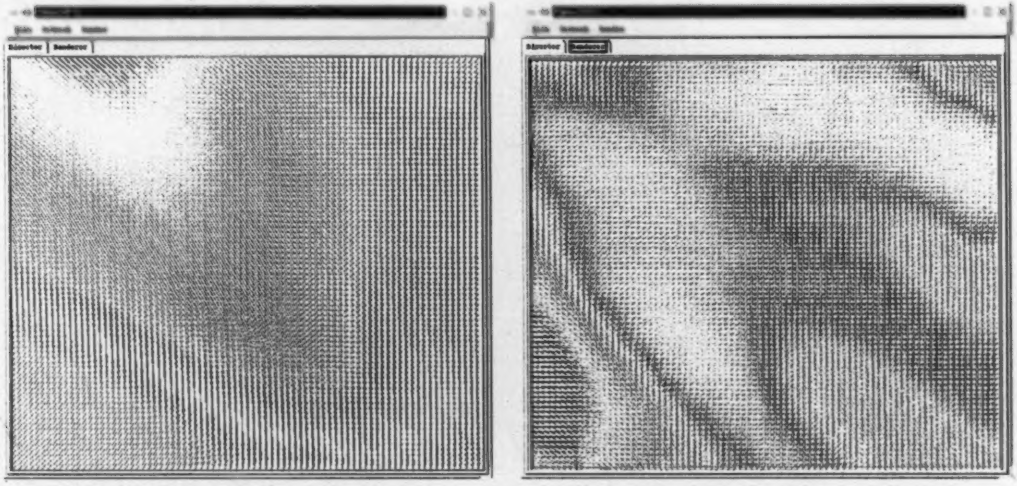
Figure 8.8: The color sequence editor.

flected by the texture and allow relationships between this data and the data variable depicted by texture to be determined.

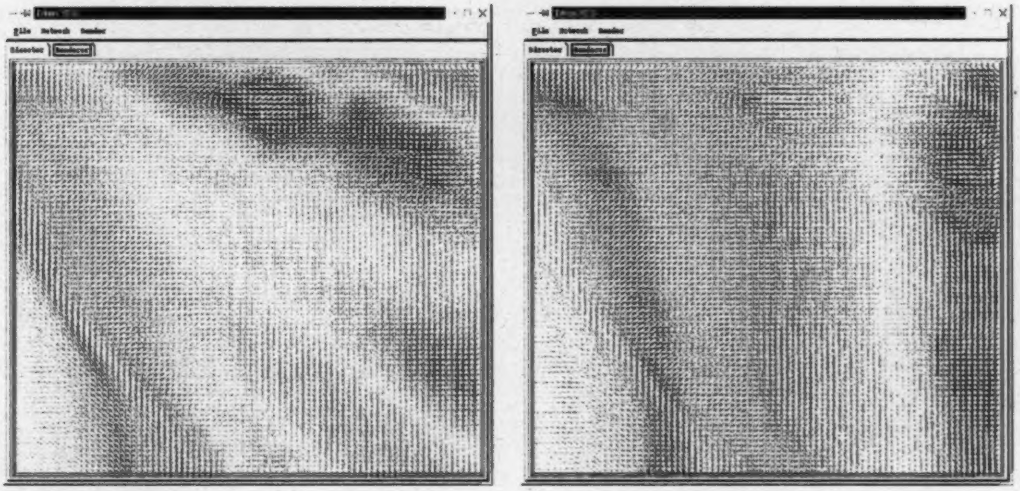
8.4 Multi-glyphs and Multi-layers

The power and flexibility of Eikon can be further illustrated by showing how it can emulate multi-glyph and multi-layer techniques similar to those proposed by Crawfis [craw91] and Laidlaw [laid98, laid99]. Multiple glyphs can easily be used to render the data by "piping" the data to be displayed to different glyphs. Each glyph can then be bound to the data present and used to render the data. Combining geometry in this way can be useful as the new "multi-glyph" may highlight features not obvious with a single glyph but as with the face glyph the increase in the complexity of each glyph can reduce the amount of data variation that can been seen. Both Crawfis and Laidlaw realize this problem in their system and both suggest reducing the visual impact of one or more glyphs to increase the impact of others. Eikon also supports this notion of layering. One method of layering is to adjust the color sequence used to render the data with a particular glyph. Figure 8.10 shows an example where the profile glyph and the line glyph have been used to render an extract from the data. To highlight the line glyph representation the profile glyphs are rendered using a light gray color sequence.

This allows the individual profile glyphs to be seen and does not obscure the



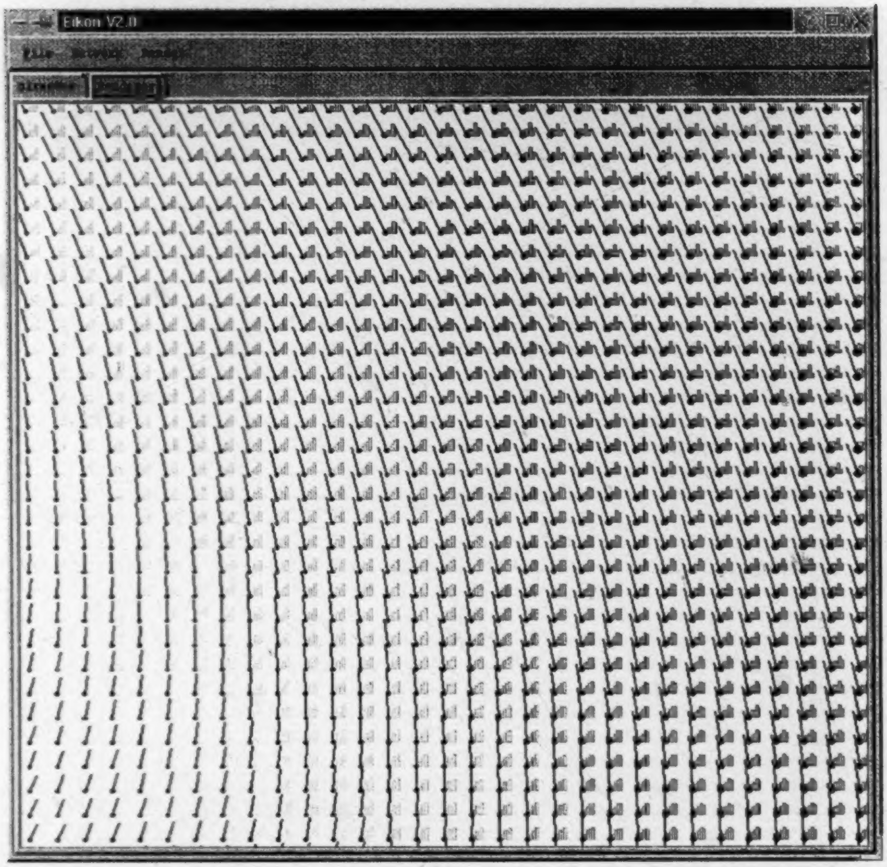
(a) Velocity (b) Vorticity



(c) Height (d) Pressure

The line glyph (a) and stick figure glyph (b,c,d) are used to display the data. The line glyph is colored using Velocity in (a). In the remaining the stick figure main body is bound to vorticity and each glyph is colored with (b) Vorticity, (c) Height, and (d) Pressure. The displayed region corresponds to an 80x80 area extracted from position (542,512) in the original dataset.

Figure 8.9: Combining color with (stick figure) glyph variation.



The output of an extracted data region is represented using a combination of the profile and line glyph. To increase the visibility of the line glyph the profile glyph is rendered using a color sequence containing light gray tones.

Figure 8.10: Multi-layered. multi-glyph image using the profile and line glyph

pattern represented by the line glyphs. Figure 8.11 shows a similar image however this time the line glyphs themselves are also colored, using the height data variable as an index into a color sequence varying in hue.

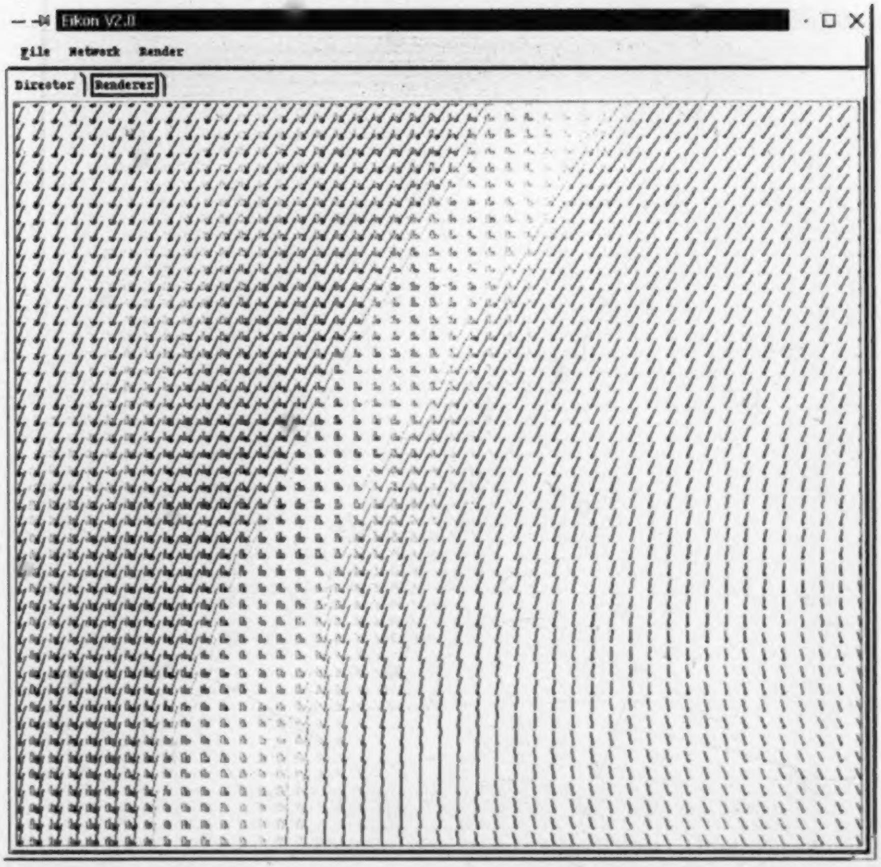
Eikon also supports another method of producing different layers. In the previous examples glyphs where applied to the same data region. Using the visual programming environment it is easy to extract different regions from the data. Each of these regions can be rendered with a different glyph. The effect of this (shown in Figure 8.12) is that when regions do not overlap they are rendered with one glyph and when they do overlap the overlapping area is rendered with both glyphs. The clutter of overlapping areas can be reduced by adjusting the visual impact of unimportant layers as in the previous section. One problem however is that the visual adjustment is applied to the entire layer and not just the overlapping area. This problem can be circumvented by extracting the regions and compositing them together before they are rendered. In this way the area of overlap can be identified and used to effect the color of glyphs. In Laidlaw's work he also indicated that clutter can be reduced by placing glyphs more intelligently in the display. Eikon can also produce images in this way using the abstraction and importance methods which will be discussed in the next two chapters.

8.5 Discussion

In the previous sections I presented several examples of how Eikon produces small scale elementary views of the data. I showed how visualization networks can be constructed that render extracts from the data, how different glyphs are used and how the binding between data variables and glyph parameters is defined. I also showed how color sequences are created using the color sequence editor, how these sequences are bound to different data variables, and how multi-glyph and multi-layer images of the data can be produced.

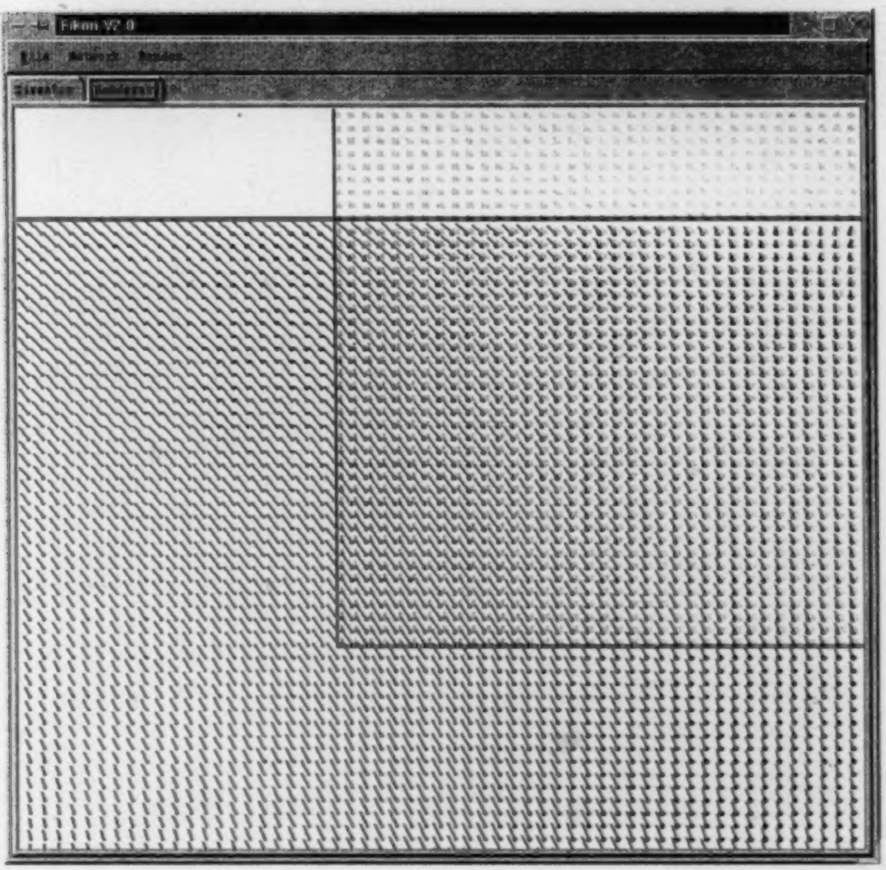
8.5.1 Advantages

Overall the ability to generate images in Eikon that display relationships between multiple variables at an elementary level was considered useful by the case study



The output of an extracted data region represented using a combination of the profile and line glyph. The line glyph is colored using the height data variable as an index into a color sequence varying in hue. The profile glyphs are rendered using a color sequence based on light gray tones. This improves the visibility of the line glyphs.

Figure 8.11: Colored multi-layered, multi-glyph image using the profile and line glyph.



In this example two extracted regions are rendered using two different glyphs (the line and the profile glyph.) Where the layers overlap the data in the overlapping areas is rendered using a combination of each glyph.

Figure 8.12: Overlapping multi-layered, multi-glyph image using the profile and line glyph.

researchers. Previously relationships between several data variables required several images depicting data variation through color to be compared side by side. Geometric coding allowed such comparisons to be made using a single image.

The researchers felt that the ability to easily change the glyph, and the binding between data variables and glyph parameters was useful as as it allowed different glyphs to be easily used for different purposes. The ability to import, select, and display data using glyphs and color within the extent of a single system was listed as a major advantage. In this respect the visual programming interface in Eikon was very well accepted. The simple point and click metaphor allowed different glyphs to be used to display the data and different data-glyph bindings to be easily defined. The pan and zoom interaction method in the renderer allowed the generated images to be readily explored at this scale also. The researchers found it much easier in Eikon to design and associate color sequences with data variables through the color sequence editor and glyph tool configuration dialog. Previously images of the data were constructed using several individual packages. This involved writing a custom program to make color images from the binary data by assigning particular color map values to data values. Individual color maps were created using a color specification tool that required colors to be specified one at a time and the final colored image required a special viewing program to be displayed. If they wished to view the data differently then new output must be generated and new color maps defined. This process was regarded as a tedious and difficult way to generate effective images of the data. It was felt that the ability to produce different views of the data quickly and easily in one system greatly enhanced the insight that could be gained about the data.

Although designed with programmers in mind, the researchers found the visual programming interface easy to use. The interface was regarded as well thought out, easy to use and required a minimum of instruction to get useful output. The ease of use can be attributed to the fact that basic networks in Eikon simply consist of import, select, and display steps and that several example networks were provided from which new networks could be derived. It should also be noted that the case study researchers were also versed in both 'C' and FORTRAN programming and may not classify as non-programmers. Also although the results presented in this chapter can be obtained with other visualization systems few existing visualization systems provide both a high level visual editor type interface which provides access to all data variables in each tool and access to the source code on which the system is based. Systems such as AVS provide a high level programming environment but do not provide access to source and as such it is difficult to customize such systems at a low level. By hiding the low level implementation details a certain amount of trust is required in using algorithms where the implementation details are not provided. On the other hand systems such as VTK provide a low level programming environment where source code is available and can be customized as needed. However without a higher level interface such systems are more difficult to use initially and more difficult to understand by non-programmers. Eikon provides both a high and low level programming interface and the source code is also available, thereby combining the advantages of both. Although Eikon is limited in extent to structured points datasets its system architecture is similar enough to VTK that the visual programming environment can be implement for this general purpose visualization system also.

8.5.2 Disadvantages

Although geometric coding was regarded as beneficial there were some restrictions noted. The researchers indicated that data should be represented by an appropriate geometric form, that is, not all glyphs were useful in every situation. The best glyphs allowed the data variables and the necessary relationships between them to be easily read. The least favored glyph was the face glyph as it was too complex and it was difficult to determine the relationship between facial properties and the data variables they represented. The line glyph was regarded as more effective as it had a simple form and there was a direct correspondence between the glyph parameters (orientation and length) and data variables such as flow magnitude and direction.

Various glyphs were deemed useful for different tasks. In some instances, for example illustrating the results to others, simple glyphs like the line glyph or profile glyph were effective as they clearly depict the information in the data. In other contexts, such as exploring the data or "data mining", the stick figure was more useful as it highlighted different patterns and structures in the data.

The researchers also noted that even though geometric coding and color coding could be used together to depict characteristics of the data, color could interfere with the visibility of geometric variation of glyphs. For example, it is difficult to determine the shape of yellow glyphs on a white background. The color sequence editor in Eikon allowed sequences to be designed quickly, however color sequences still had to be designed with care. The ability to readily change the color sequence provides some support for rectifying this problem but a better solution would be to have several predefined color sequences for specific purposes, or provide guidelines for constructing good sequences.

The researchers also voiced concerns about the balance between flexibility and ease of use within Eikon. One concern in particular related to tools for reading different data formats. It was surmised that as data formats change it would be difficult to import the new data into Eikon. This is a concern that affects Eikon and most systems with high level visual programming environments. From the outset, however, Eikon has been considered a tool used and supported through collaboration with a visualization expert. Therefore, as new requirements are encountered these are implemented by the visualization expert within Eikon. Eikon has been designed to make such extensions relatively straightforward.

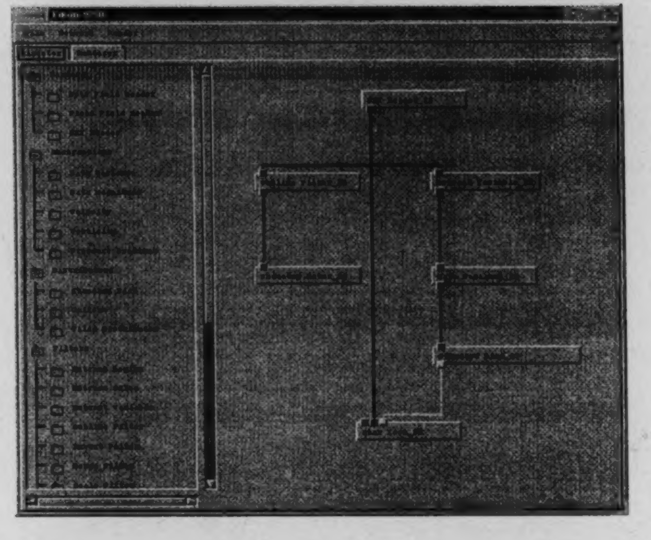
Finally, in the context of this case study, elementary views were only useful in as much as these areas could be identified and displayed from larger scale overviews of the data. That is, before examining details it is necessary to look at broad overviews first. In the next chapter I describe how this can be accomplished in Eikon.

Chapter 9

Case Study - Global Views

The elementary views in the last chapter represent only a fraction of 1% of the total number of data points in the GFD dataset. In contrast global views display images relating to all data points in the datasets. Global views are important as they illustrate answers to global questions pertaining to all data points. Arguably visualization is most effective in these cases as images can be constructed that represent structures and patterns in the data that would not easily be detected by other means.

Global overviews play a crucial role in understanding and developing the mathematical model for the GFD simulation. The researchers in Mathematics are more interested in large scale features generated by the simulation than in precise details. The main advantage of global overviews, therefore, is the ability to see particular features in the dataset at a large scale and compare the location, size, shape, and other characteristics of these features with results from other simulations or real world measurements. Global overviews also allow the evolution of features over time to be visualized. The current simulation consists of 200 time slices and global overviews allow the evolution of features over time again to be compared with real world measurements and with expected behavior. Global overviews also depict the location of interesting features and provide the basis for exploring the data in more detail, i.e, they illustrate features in the data at a large scale and provide a "map" as to which areas may warrant further investigations.



This example network reads the GFD dataset and extracts the height variable from all data points. A variance importance function is applied to this variable and is used to determine a multi-scale distribution structure. This structure is used to distribute star glyphs over the image.

Figure 9.1: A visualization network using importance.

9.1 Global Overviews in Eikon

In Eikon global overviews are created by using the abstraction, importance, and distribution method specified in Chapter 5. The current GFD dataset is composed of 200 time slices each with over one million data points. Displaying the data by associating a glyph which each data point affords each glyph one pixel or less on a typical computer display. At this scale no geometric variation is possible resulting in images that do not emphasize any of the data and that can be time consuming to produce.

Using appropriate abstraction and importance functions, significant features of the data can be determined. Combined with the distribution process, images can be produced that highlight important features through the spatial distribution of glyphs over the display. The distribution process also allows variable numbers of glyphs to be used thereby reducing the problem of rendering times for large datasets.

9.1.1 Importance Functions

Figure 9.1 shows an example of a visualization network that creates an overview of a slice from the GFD dataset. Similar to the elementary views, this network is constructed by selecting the appropriate tools for reading, transforming, and displaying the data and connecting them to form a visualization network. Unlike elementary views, the network in Figure 9.1 also uses an importance function and a glyph distributor to determine the placement of glyphs in the generated image. Rather than extract a small region of the dataset, an importance function is applied to a variable extracted from each data point in the entire dataset. In this example the height variable is extracted and a variance importance function is applied to it. The objective of the variance importance function is to highlight structure or form within the data by indicating areas of high variance or edges. Equation 9.1 describes how the variance of a data point is computed as the average of the square of the differences between the data point and its eight connected neighbors. Note at the edges of the dataset the data point will have less than eight connected neighbors and the computation is adjusted accordingly.

$$Variance(x,y) = \frac{\sum_{i=-1}^{i=1}\sum_{j=-1}^{j=1}(f(x,y) - f(x+i,y+j))^2}{NumberofConnectedNeighbours} \tag{9.1}$$

The output of this network is displayed in Figure 9.3, where 70,000 star glyphs have been distributed over the image based on the output variance importance function. Each of the star points are bound to one of the five variables in the dataset as described in Table 9.1. Each glyph is also colored using the value of height as an index into a color sequence (Figure 9.2) varying in hue. As can be seen the distribution clearly indicates the global variation in height over the entire dataset.

The focus of the rendered image can be easily changed by changing the variables to which the importance function is applied and/or by changing the importance function used. Eikon contains an example of a second importance function that determines areas where the magnitude of a particular variable, or set of variables is large from other areas where it is small. If only a single variable is presented to the magnitude importance function then the magnitude of particular data values is simply determined as its distance from the smallest value associated with the data variable across

Glyph Parameter	Minimum Value	Maximum Value	Data Variable
Star Point 1	0.0	2.0	Height
Star Point 2	0.0	2.0	Pressure
Star Point 3	0.0	2.0	X Velocity
Star Point 4	0.0	2.0	Y Velocity
Star Point 5	0.0	2.0	Vorticity
Color	0.0	1.0	Height

Table 9.1: The binding assigned to the star glyph (global view).

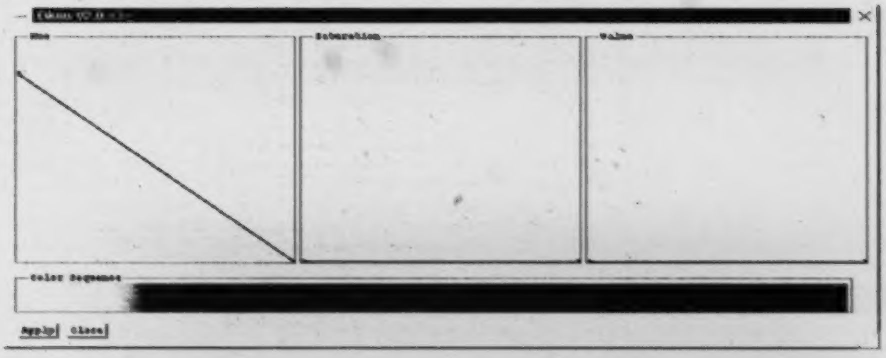
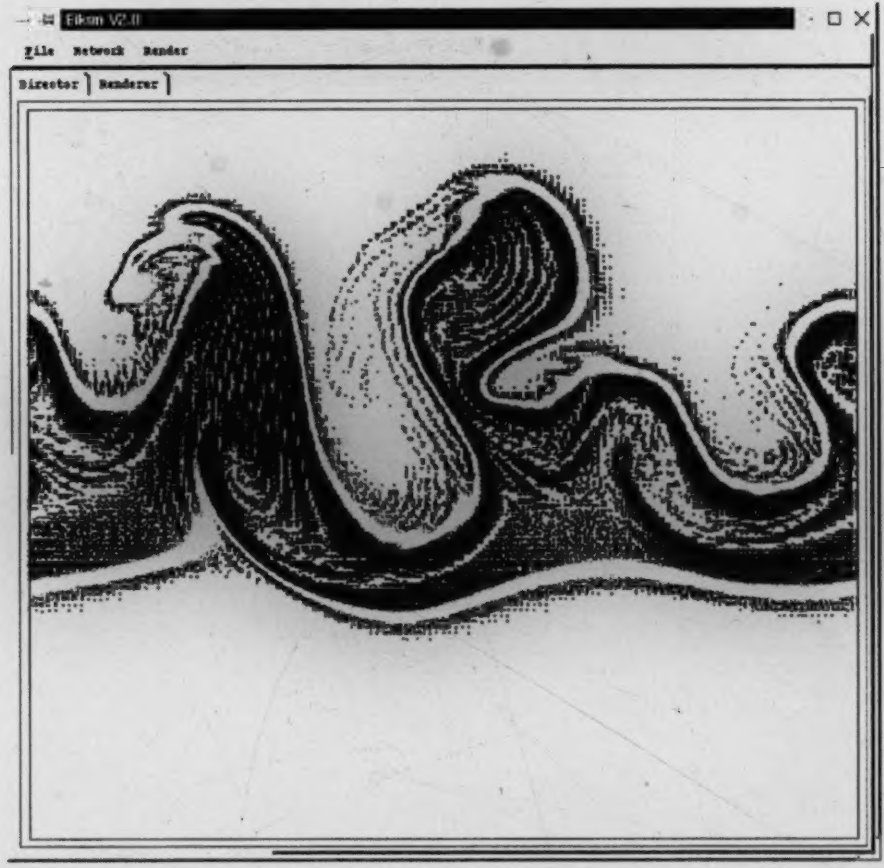


Figure 9.2: The color scale used in the height variation global view



A global overview focusing on variation in height over the dataset. 70,000 star glyphs are used and each glyph is colored using the height variable.

Figure 9.3: Height variation overview.

Glyph Parameter	Minimum Value	Maximum Value	Data Variable
Bar 1	0.0	2.0	Height
Bar 2	0.0	2.0	Pressure
Bar 3	0.0	2.0	X Velocity
Bar 4	0.0	2.0	Y Velocity
Bar 5	0.0	2.0	Vorticity
Color	0.0	1.0	Pressure

Table 9.2: The bindings assigned to the profile glyph (global view).

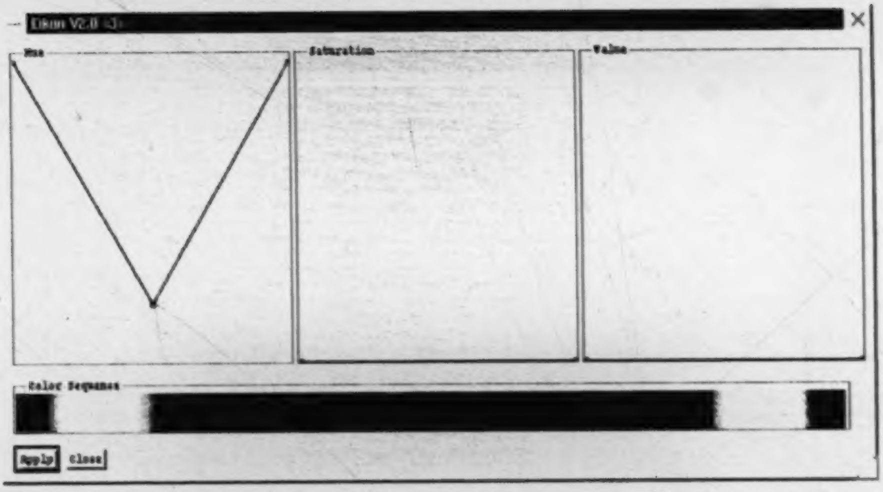
all data points. If more than one data variable is presented the magnitude is computed as the magnitude of the vector defined by the data variables as illustrated in Equation 9.2.

$$Magnitude = \sqrt{\sum_{i=1}^{i=N} C_i^2} \tag{9.2}$$

Where N is the number of components in each data point and C_i represents the ith component of a particular data point.

Figure 9.5 shows the output of the network in Figure 9.1 where the variance function has been replaced with a magnitude importance function, the magnitude function is applied to the pressure data variable, a profile glyph is used to display the data, and the pressure values are used to determine the color of each glyph. In this case 50,000 profile glyphs have been used to display the data with a data-glyph binding described in Table 9.2. A color sequence varying in hue as illustrated in Figure 9.4 is used to illustrate the pressure variable also. In Figure 9.5 the magnitude of pressure is evident over the image. Three areas of high pressure can be determined and three areas of low pressure are also visible.

In Figure 9.3 and Figure 9.5 each glyph was colored using the same variable on which the distribution was based. This has the effect of reinforcing the structures depicted by the distribution. At this scale, no geometric variation is visible through individual glyphs so no other information is visible in the image. Using the glyph configuration dialog and the color sequence editor, color sequences can be designed and bound to other variables from the dataset. Figure 9.6 and Figure 9.7 show several examples where color is used to represent a variable other than that represented



A color sequence varying in hue is used to depict the pressure in the global overview based on pressure magnitude. A "V" shaped function is used to illustrate high, medium, and low pressure.

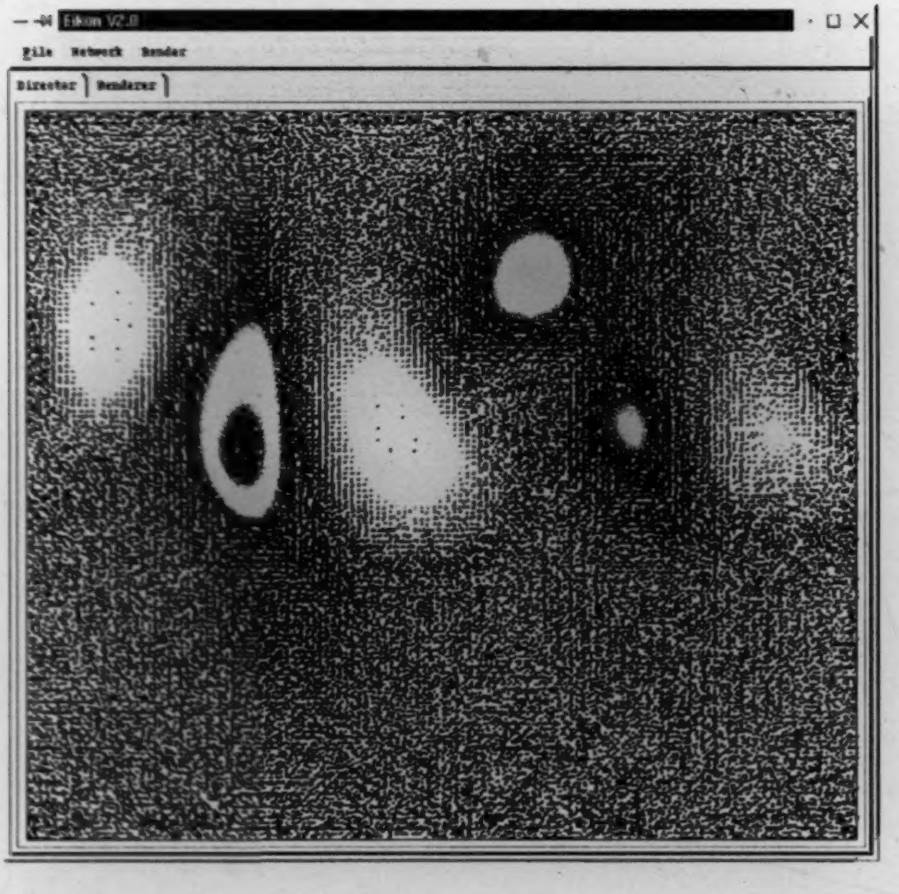
Figure 9.4: The color sequenced used for pressure magnitude (global view.)

through glyph distribution.

These images are constructed by using the glyph configuration dialog to define a color sequence and binding this sequence to any of the data variables in the dataset. Global overviews which represent multiple characteristics of the data are possible where the data variable represented by glyph density can be related to another data variable encoded using glyph color. As in the previous views where color was used to generate elementary views, color must be used with care. In some images (such as Figure 9.6(a) showing variation in height along with pressure) the variation in glyph density is unaffected by the use of color. In other images (such as Figure 9.7(a) showing magnitude of pressure along with Y-velocity) the color has the tendency to mask the data represented through glyph distribution. The perceived density of glyphs is dependent on the visibility of individual glyphs and as color affects the visibility of glyphs this can interfere with the perceived glyph density.

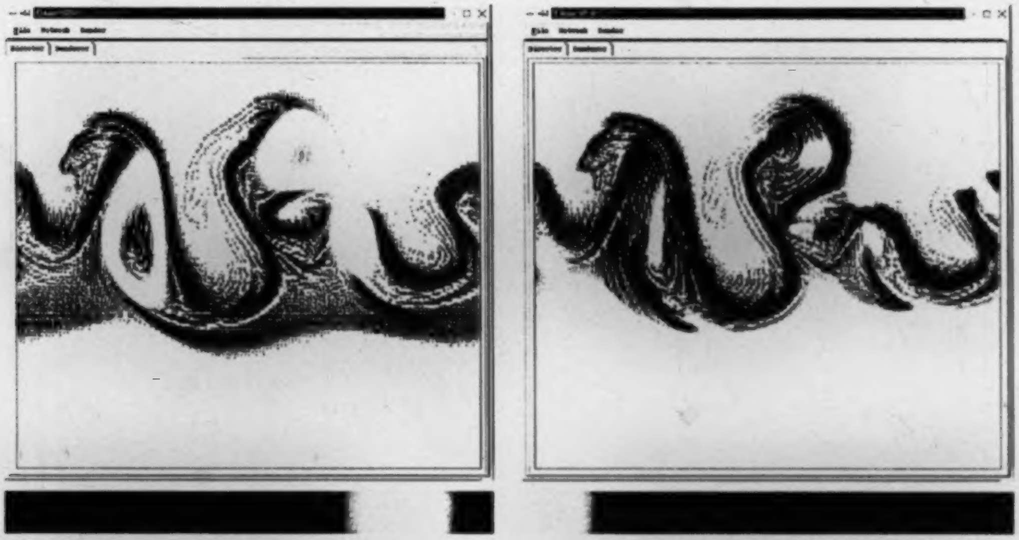
9.1.2 Abstraction Functions

In the above examples variance and magnitude importance functions where applied to data variables representing height and pressure. These correspond to the two variables

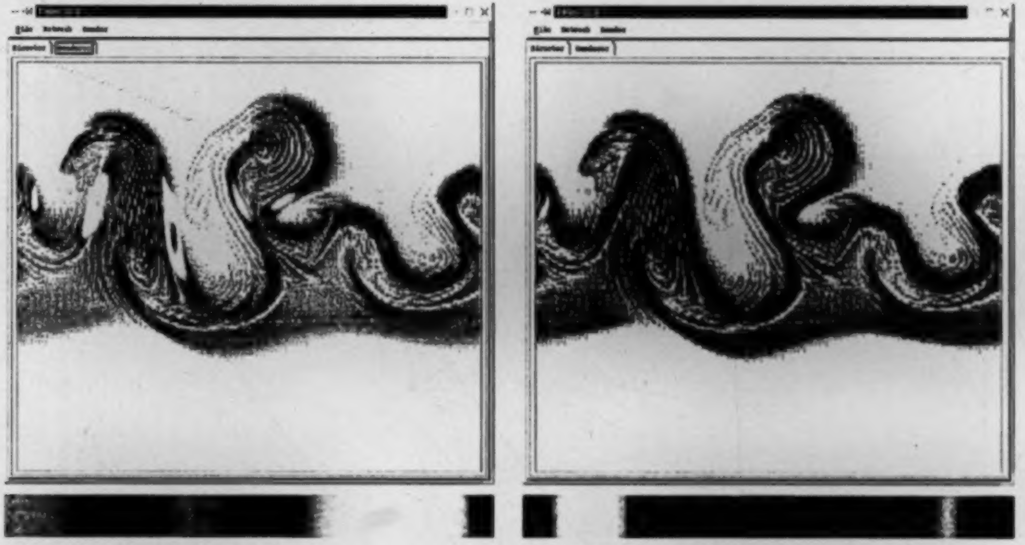


This image shows a global overview focusing on magnitude of pressure over the dataset. 50,000 profile glyphs are used where each glyph is also colored using the pressure variable.

Figure 9.5: A global view of pressure magnitude.



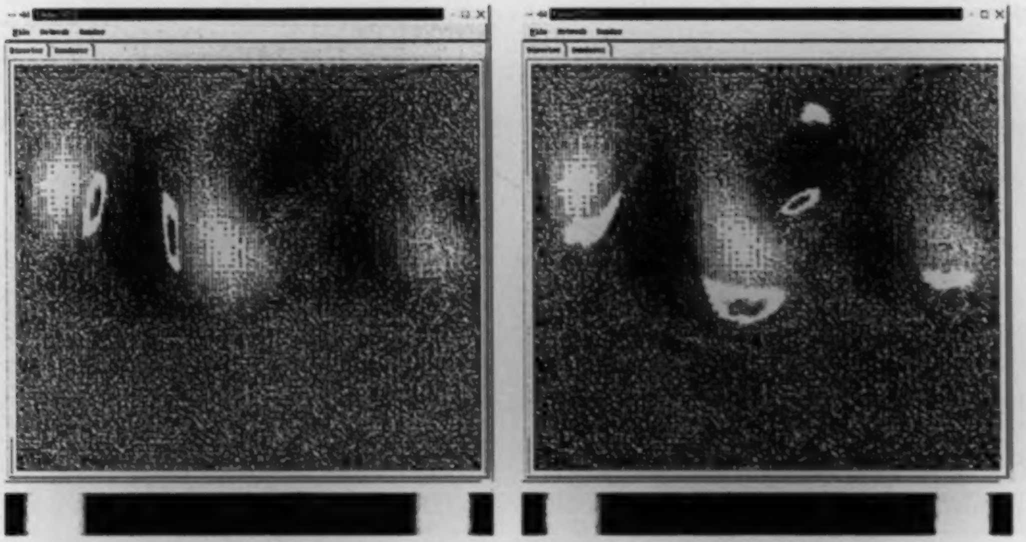
(a) Height and Pressure (b) Height and Velocity



(c) Height and Velocity (d) Height and Vorticity

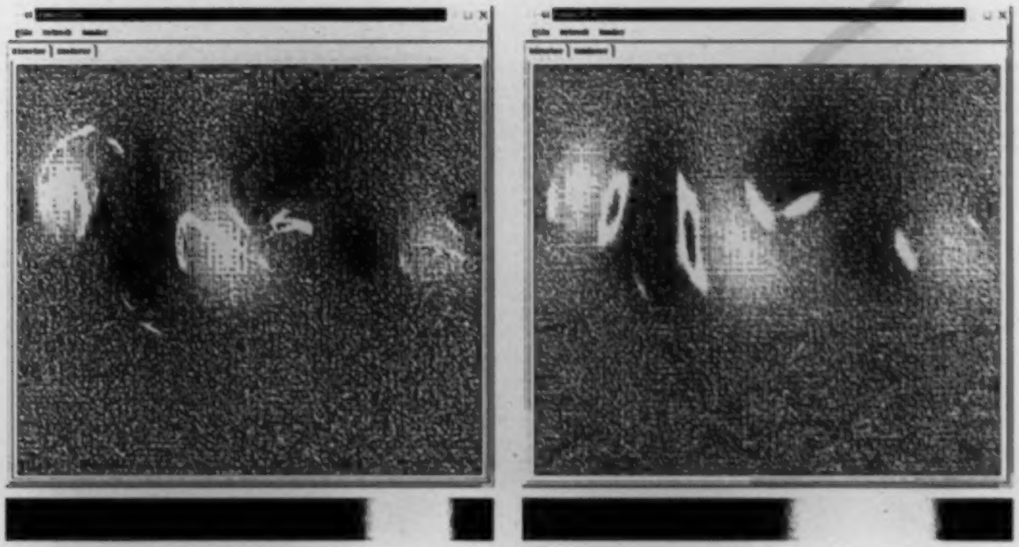
In these images the height is illustrated through glyph distribution and a selection of other data variables through color. In each instance the color sequence is illustrated beneath the image.

Figure 9.6: Examples of glyph distribution (height) and color in global overviews.



(a) Pressure and Y Velocity

(b) Pressure and X Velocity



(c) Pressure and Vorticity

(d) Pressure and Velocity Magnitude

In these images the pressure magnitude is illustrated through glyph distribution and a selection of other data variables through color.

Figure 9.7: Examples of glyph distribution (pressure) and color in global overviews.

computed by the fluid simulation code. Other data of interest, and correspondingly other views of the data, can be generated by computing abstractions from the original data. In this case study two other abstractions where implemented that determine velocity and vorticity from the pressure data output by the simulation.[1]

Velocity Abstraction

In Eikon velocity is computed from the pressure values provided in the GFD dataset as described below:

$$V_x = -\frac{\partial p}{\partial y} \tag{9.3}$$

$$V_y = \frac{\partial p}{\partial x} \tag{9.4}$$

$$V_z = 0.0 \tag{9.5}$$

In this equation, V_x, V_y and V_z are the x, y and z components of the velocity vector and p is pressure. The GFD dataset is a two-dimensional dataset and therefore the resulting flow field is also two-dimensional (the z component is zero).

Vorticity Abstraction

The vorticity of a vector field is a method of describing the rotation that exists within the vector field. Vorticity is described by a vector where the direction of the vector indicates the axis of rotation and the magnitude of the vector indicates the amount of rotation [stew99]. We use the following equations to determine the three components of the vorticity vector:

$$\Omega_x = 0.5 \times (\frac{\partial V_z}{\partial y} - \frac{\partial V_y}{\partial z}) \tag{9.6}$$

---

[1] The previous examples use a DEX reader which generates height, pressure, velocity, and vorticity data. This reader is implemented as a tool that encapsulates tools for reading the GFD data and for computing the velocity and vorticity abstractions from it.

Glyph Parameter	Minimum Value	Maximum Value	Data Variable
Star Point 1	0.0	2.0	Height
Star Point 2	0.0	2.0	Pressure
Star Point 3	0.0	2.0	X Velocity
Star Point 4	0.0	2.0	Y Velocity
Star Point 5	0.0	2.0	Vorticity
Color	0.0	1.0	Velocity Magnitude

Table 9.3: The bindings assigned to the star glyph (global view).

$$\boldsymbol{\Omega_y} = 0.5 \times (\frac{\partial V_x}{\partial z} - \frac{\partial V_z}{\partial x}) \tag{9.7}$$

$$\boldsymbol{\Omega_z} = 0.5 \times (\frac{\partial V_y}{\partial x} - \frac{\partial V_x}{\partial y}) \tag{9.8}$$

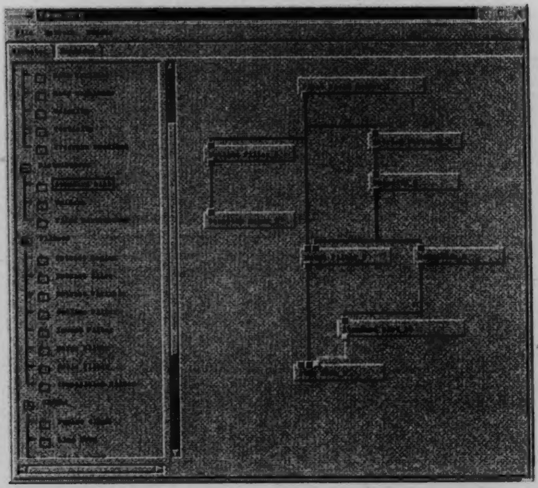
In this equation, $\boldsymbol{\Omega_x}$, $\boldsymbol{\Omega_y}$, and $\boldsymbol{\Omega_z}$ represent the x, y and z components of the vorticity and V_x, V_y and V_z represents the x, y, and z components of the velocity vector.

As the velocity field is a two-dimensional vector field, the vorticity is represented by a single scalar value. The axis of rotation in this field is perpendicular to the two dimensions of the field, i.e., the axis of rotation points in the z-axis direction and the computed scalar value represents the amount of rotation around this axis at each point.

9.1.3 Velocity and Vorticity Views

Figure 9.8 presents an example of a visualization network that uses the velocity abstraction and a magnitude importance function to display the data. The computed velocity is also merged with the original data and the distributor uses velocity magnitude to distribute star glyphs over the image. In this network 30,000 star glyphs are displayed using the binding described in Table 9.3. The output of the network is shown in Figure 9.9

The image generated can easily be changed by changing the abstraction, or by linking abstractions together (computing abstractions of abstractions). Figure 9.10

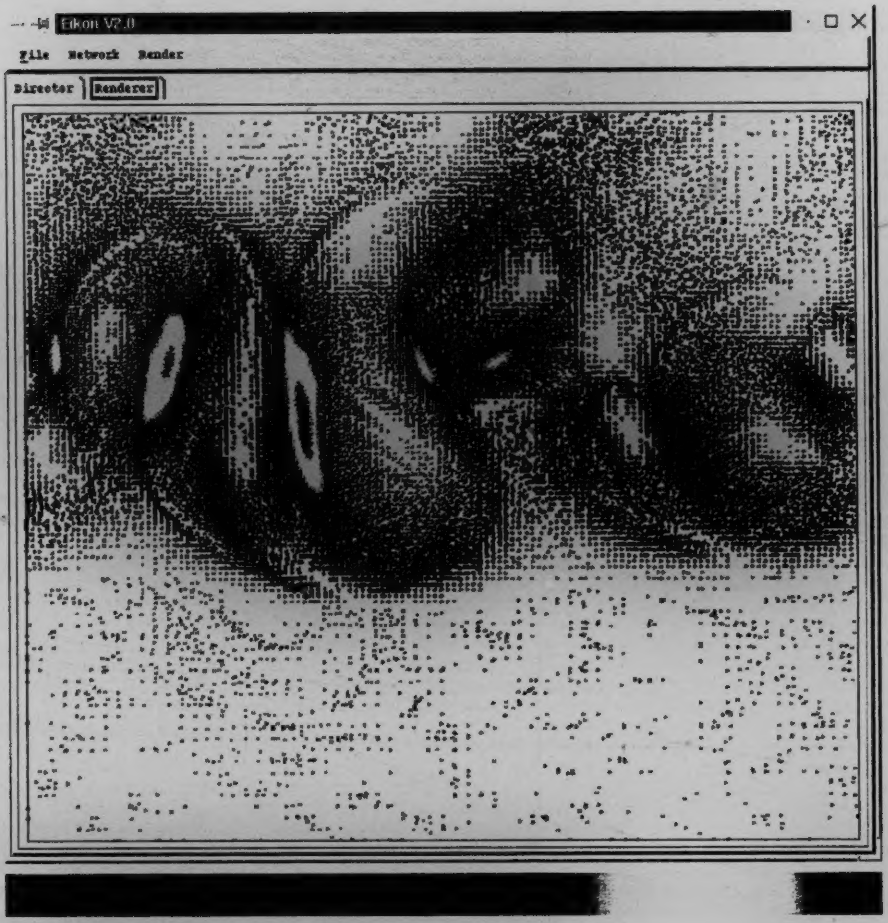


This example network reads the GFD dataset and applies a velocity abstraction to the pressure component. The magnitude of the velocity is used to distribute 30,000 star glyphs over the image.

Figure 9.8: A network using abstraction and importance.

Glyph Parameter	Minimum Value	Maximum Value	Data Variable
Bar 1	0.0	2.0	Height
Bar 2	0.0	2.0	Pressure
Bar 3	0.0	2.0	X Velocity
Bar 4	0.0	2.0	Y Velocity
Bar 5	0.0	2.0	Vorticity
Profile Size	0.0	4.0	Unbound
Color	0.0	1.0	Vorticity

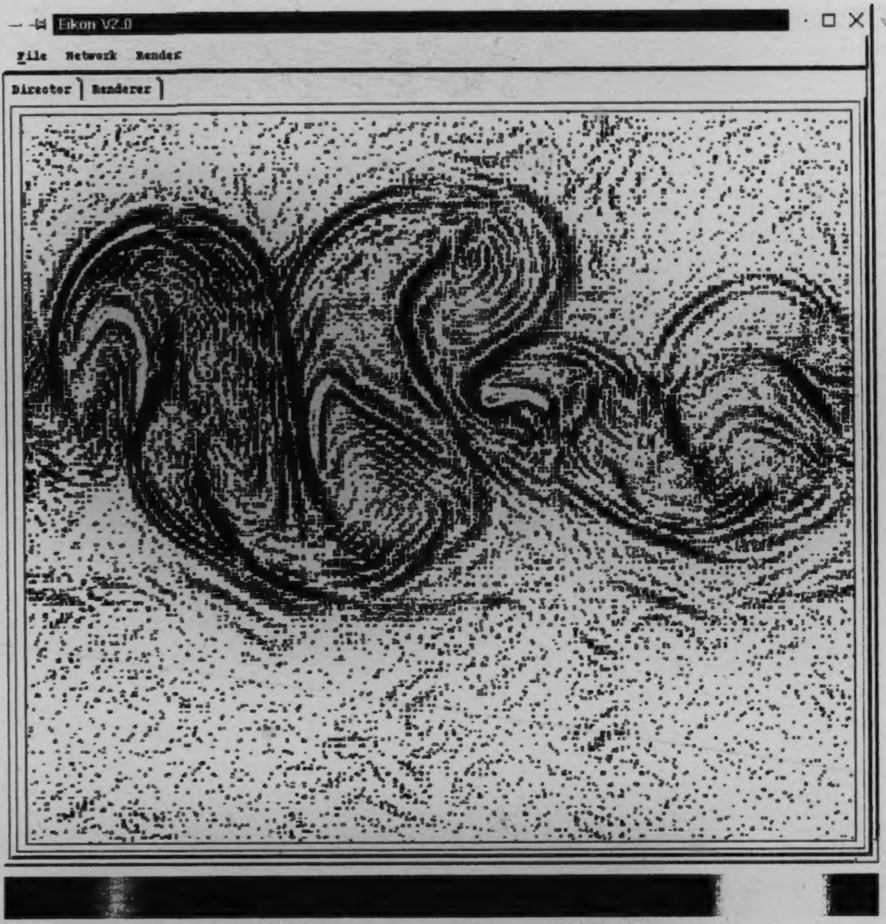
Table 9.4: The bindings assigned to the profile glyph (global view).



Velocity Magnitude.

In this image velocity magnitude is illustrated through the distribution of 30,000 star glyphs over the image. The color of each glyph indicates velocity magnitude also.

Figure 9.9: Velocity Magnitude.



Vorticity.

In this image vorticity variance is illustrated through the distribution of 50,000 profile glyphs over the image. The color of each glyph also indicates vorticity.

Figure 9.10: Vorticity Variance.

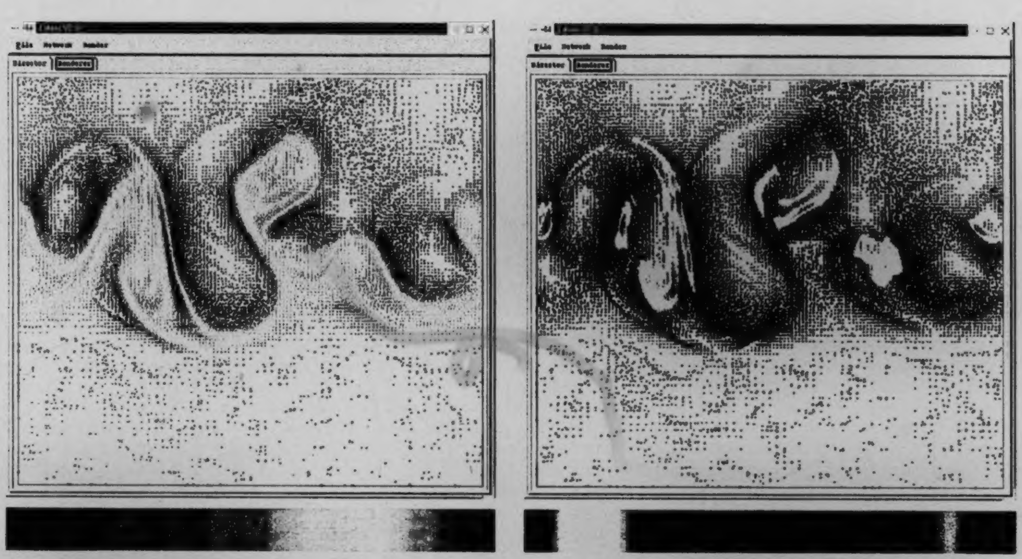
shows the output of applying a velocity abstraction to compute velocity vectors and a vorticity abstraction to compute the vorticity of the velocity field. The output of both abstractions are merged with the original dataset and displayed using a variance importance function applied to the vorticity component. 50,000 profile glyphs are distributed based on vorticity using the binding shown in Table 9.4. Finally Figure 9.11 presents some examples of using the velocity and vorticity abstractions where color is used to depict other variables from the data.

9.1.4 Multi-layer Views

In a similar manner to Laidlaw's technique [laid99] multi-layer views can be constructed within Eikon. Figure 9.12 presents an example in which two abstraction functions are rendered as two distinct layers. The first layer represents a distribution based on the variance in the height variable over the entire dataset. This layer is rendered in a dark gray color and represents areas within the dataset where changes in height are large. On top of this layer a second layer, representing velocity magnitude, is displayed. This layer is also colored using the velocity magnitude as an index into a color sequence varying in hue. As Laidlaw describes in his paper the first layer (height variance) provides a "backdrop" against which the second layer (velocity magnitude) can be viewed. At this scale little variation in individual glyphs is visible and both layers are rendered using 30,000 square glyphs.

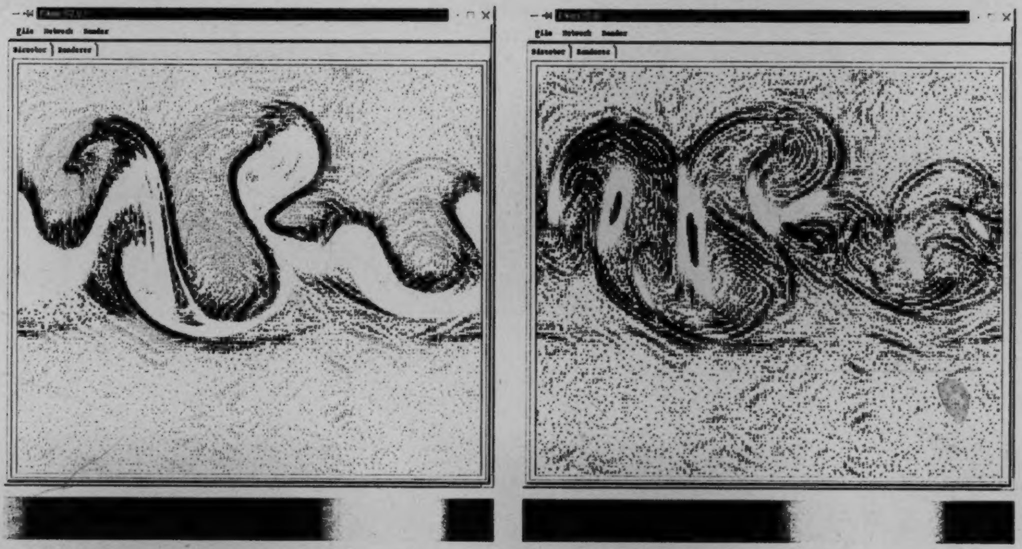
9.2 Level of Detail

The purpose of distributing glyphs over the image is to allow particular characteristics of the data to be determined at a global level. Important areas can be determined through dense areas of glyphs in the image and by changing the importance function, the abstraction function, or how color is used to render each glyph several different global overviews can be created. Another role of global overviews, however, is to indicate areas in the data that may require further exploration. In Eikon these areas are explored by interactively panning and zooming the data displayed. In the previous examples the data has been displayed using between 30,000 and 70,000 glyphs. In such images interactive data exploration is only possible on computers with powerful



(a) Velocity and Height

(b) Velocity and Vorticity

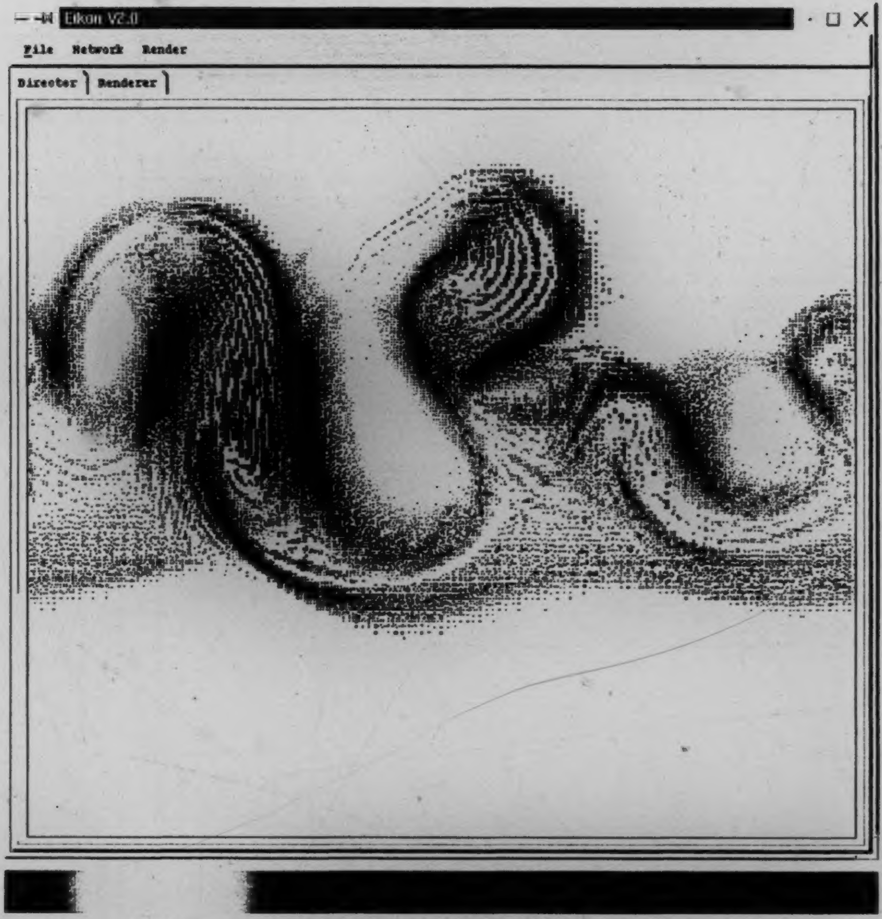


(c) Vorticity and Height

(d) Vorticity and Velocity

These images present examples of the data rendered using velocity and vorticity abstractions and color to represent other data characteristics.

Figure 9.11: Velocity and vorticity abstractions using color.



Velocity (Colored) and Height as different layers

The above image presents an example of rendering different layers using different abstraction functions. One abstraction function (height variance) is used as a base layer on top of which a second abstraction function (velocity magnitude) is displayed.

Figure 9.12: Multi-layered image using two abstraction functions.

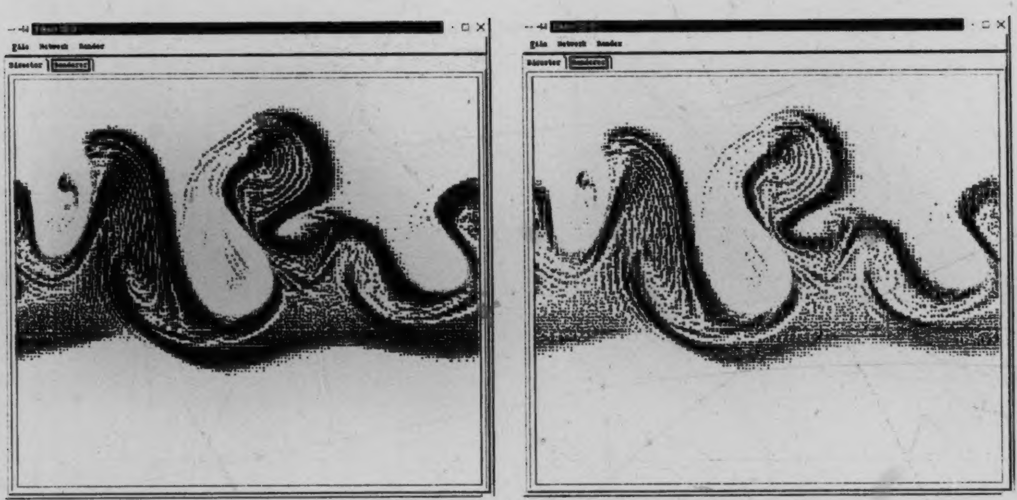
graphics hardware. Often such hardware is not available to the researchers who wish to explore the data they generate. In this respect it is necessary to adjust the amount of rendering that is required to produce each image while maintaining the information that is displayed.

Using a simple glyph is a straight forward method of reducing the rendering load. In the global overviews presented so far no geometric variation is visible on a per glyph basis and as a result using a simpler glyph, such as a square glyph, will generate similar images at this scale. The glyph distribution method in Eikon offers another method of reducing the rendering load. The distribution process ensures that areas receive a number of glyphs proportional to their relative importance to other areas. Therefore as the number of glyphs distributed is reduced important areas will receive relatively more glyphs than unimportant areas, thereby maintaining the significant characteristics of important features. Figure 9.13 shows four examples of using different numbers of square glyphs to display variation in the height variable. Figure 9.13(a) uses 70,000 glyphs, Figure 9.13(b) uses 50,000 glyphs, Figure 9.13(c) uses 20,000, and Figure 9.13(d) uses 5,000 glyphs. In each case the glyphs are colored using the pressure data variable.

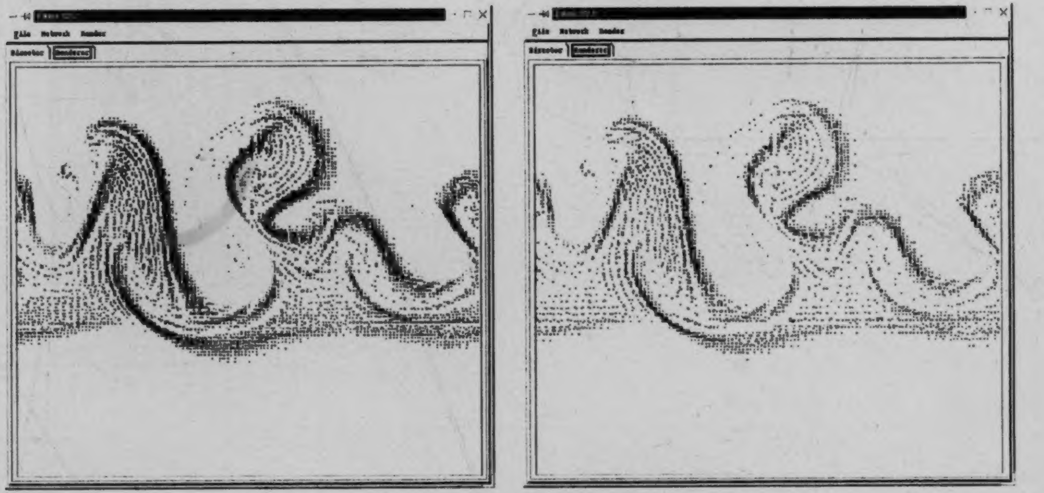
Table 9.5 presents the frame rates achieved using different glyphs and different numbers of glyphs on three different computer platforms. The machines used correspond to an SGI Onyx2, an SGI Octane, and a Intel Katmai PC. The Onyx2 uses the IRIX 6.5 operating system and contains six 195 Mhz MIPS R10000 CPUs each with a MIPS R10010 floating point unit, 768 Mbytes of main memory, and three Infinite Reality 2 graphics boards[2]. The Octane also uses the IRIX 6.5 operating system and contains two 175 MIPS R10000 CPUs each with a MIPS R10010 floating point unit, 128 Mbytes of main memory, and an SI (with texture option) graphics board. The Intel Katmai uses the Linux operating system (Mandrake 7.1 distribution) and contains, an Intel Pentium III 450Mhz CPU, 320 Mbytes of main memory, and an nVidia RivaTNT2 graphics board. The images shown in Figure 9.13 indicate that a lot of information can be displayed when only a few thousand glyphs are rendered and the frame rates in Table 9.5 indicate that by varying the number and type of glyphs used to display the data interactive frame rates can be achieved on a wide

---

[2] Only one of the graphics boards was used



(a) 50,000 Glyphs (b) 20,000 Glyphs



(c) 10,000 Glyphs (d) 5,000 Glyphs

Variation in height is used to distribute a varying number of glyphs over the image. Each glyph is colored using the pressure variable.

Figure 9.13: Reducing global level of detail.

Star Glyph

Number of Glyphs	SGI Onyx2	SGI Octane	Intel Katmai
70,000	0.58 Frames/Sec	0.23 Frames/Sec	0.24 Frames/Sec
50,000	0.84 Frames/Sec	0.33 Frames/Sec	0.29 Frames/Sec
20,000	2.00 Frames/Sec	0.82 Frames/Sec	0.77 Frames/Sec
10,000	4.00 Frames/Sec	1.67 Frames/Sec	1.58 Frames/Sec
5,000	7.49 Frames/Sec	3.17 Frames/Sec	3.23 Frames/Sec

Profile Glyph

Number of Glyphs	SGI Onyx2	SGI Octane	Intel Katmai
70,000	2.31 Frames/Sec	0.82 Frames/Sec	0.20 Frames/Sec
50,000	3.33 Frames/Sec	1.12 Frames/Sec	0.27 Frames/Sec
20,000	7.49 Frames/Sec	2.52 Frames/Sec	0.72 Frames/Sec
10,000	15.29 Frames/Sec	5.07 Frames/Sec	1.51 Frames/Sec
5,000	29.96 Frames/Sec	10.1 Frames/Sec	2.90 Frames/Sec

Square Glyph

Number of Glyphs	SGI Onyx2	SGI Octane	Intel Katmai
70,000	5.45 Frames/Sec	1.54 Frames/Sec	1.40 Frames/Sec
50,000	7.48 Frames/Sec	1.81 Frames/Sec	2.05 Frames/Sec
20,000	15.32 Frames/Sec	4.13 Frames/Sec	4.94 Frames/Sec
10,000	29.98 Frames/Sec	10.32 Frames/Sec	9.83 Frames/Sec
5,000	60.03 Frames/Sec	18.11 Frames/Sec	18.18 Frames/Sec

Table 9.5: Frame rates for various levels of detail.

range of hardware.

When a global view of the data is zoomed individual glyphs are allocated more resolution and can represent more detail about the data. From Table 9.5 the square glyph provides the highest frame rate for panning and zooming, however, when zoomed this glyph provides no extra information about the data. Currently in Eikon this requires that a more complex glyph be used initially or the glyph be manually changed when an interesting area is explored. Using level of detail techniques from the computer graphics field, Eikon could be extended to automatically change the glyph based on the scale at which the data is viewed and the current frame rate.

A second problem must also be addressed. When overviews of the data are rendered each glyph is rendered at an increased size in world space. This is necessary to avoid aliasing problems when many glyphs are displayed. The result however is that glyphs overlap and when zoomed, glyphs may interfere with each other in a negative way. In the current system this problem is addressed by manually changing the size of each glyph when the data is zoomed. Again, however, it is possible to extend the system such that the size of individual glyphs is adjusted in world space automatically according to the current zoom level.

9.3 Discussion

In the previous section I presented how the visual programming environment in Eikon can be used to produce global overviews of large multivariate datasets using the process of abstraction, importance, and distribution to display the data. I illustrated how importance functions based on data magnitude and data variance can be used with a glyph distribution tool to produce overviews that highlight characteristics of data variables explicitly represented in the dataset. I also showed how importance functions can be combined with abstraction functions to produce views depicting characteristics derived from existing data variables. Abstraction functions corresponding to velocity and vorticity were used to illustrate this process. I also indicated how the color sequence editor can be used to design specific color sequences that can be used to show other information about the data beyond that represented through variation in glyph density.

The global overviews produced consist of images where important features are represented through variation in glyph density over the image and where relationships to other variables can be seen through color. I also show how interesting areas visible in the global overviews can be interactively explored by adjusting the number of glyphs and the type of glyph used in order to maintain interactive frame rates for panning and zooming.

9.3.1 Advantages

The ability of Eikon to display quantities in the dataset and to form and display new quantities from the existing data was regarded as advantageous by the researchers in the case study. In particular the ability to identify important characteristics of the data through abstraction and importance and to illustrate this via glyph density and color, coupled with interactive panning and zooming, was regarded as very useful. Also the ability to define whatever abstraction and importance needed provided a lot of flexibility that was important in visually exploring and understanding the data generated and the underlying mathematical model.

The ability to form global overviews that focus on characteristics of a single variable from the data was regarded as especially important when looking at the data for the first time. In such instances the researcher is interested in seeing and understanding selected features of the data in their entirety and is not interested in irrelevant details. The distribution process in Eikon was regarded as useful in this role as it generates overviews by trading detail in some areas (few glyphs placed) for a global overview using glyph density to represent important features. Depicting areas of high variation at this scale, using the variance importance function, was considered useful as this could indicate deficiencies in the numerical simulation such as numerical instabilities, or, if the variations are extreme, a break down in the assumptions that the simulation is based upon. Illustrating the magnitude of data variables, using the magnitude importance function, was also valued. For example depicting the distribution of high and low velocities is of interest to the researchers. Using abstractions to determine the physically, or otherwise, relevant components that the raw data implies was also very useful. Visualizing these components gives the researchers a deeper understanding of the data and the mathematical model. The velocity and vorticity abstractions provided were of interest as they represent physically important properties of the fluid. The researchers also indicated that other abstraction functions such as abstractions based on potential vorticity would also have been very beneficial. Potential vorticity[3] and its characteristics, for example, are important in determining if the mathematical model is behaving as expected and determining the existence of other features in the data.

Although focusing on one variable is beneficial when looking at the data at a global level it is also useful to correlate the data displayed with other data variables in the dataset. The researchers indicated, for example, that if a data variable had a high degree of variability then it was often enough to focus on just this variable. However, they also indicate that it was beneficial to represent different types of information over the same spatial domain. Within Eikon, at a global level, this ability was provided using glyph density and color. Again, the researchers found the color sequence editor a useful tool for designing color sequences. Using the editor color sequences could be easily constructed that allow color to depict other data variables that did not, in the eyes of the researchers, affect the visibility of the data displayed through glyph distribution too badly.

Considering that the global overviews do not provide a lot of detail about the data this was not regarded as a major problem by the researchers. At first glance overviews depicting a single data variable can be useful by indicating areas of interest without being burdened by potentially unnecessary detail. Also the ability to pan and zoom the displayed data allowed the researchers the ability to examine these areas in more detail. More importantly however the researchers indicated that it was desirable to go quickly between the global and detail views. In Eikon they found the ability to control the amount of detail display beneficial. The ability to reduce the number of glyphs was especially useful as it allowed interactive exploration on the computers they used - SGI O^2 and Indigo 2's, which do not have very powerful graphics hardware.

---

[3] computed using the Coriolis parameter (planetary vorticity), relative vorticity, and the fluid layer thickness

9.3.2 Disadvantages

The main disadvantage voiced by the researchers pertained to the discrete nature of the displays generated by distributing glyphs over the image. Typically in the past the researchers have used two-dimensional color images to depict variables from their datasets. Although the simulation itself generates data values at discrete locations the fluid being modeled is continuous. The researchers believed that a continuous color image depicted this property of the fluid better than images with varying numbers of glyphs at different locations.

Another problem noted by the researchers was the lack of other, more usual, visualization methods in Eikon. As the researchers are used to visualization methods such as contour plots and the previously mentioned color images the inability to produce these was considered problematic. They also indicated that the ability to combine color images, contour plots, and glyph distributions in a single visualization would help alleviate their concerns regarding the discrete nature of the glyph display and may illustrate more information about the data at global, intermediate, and elementary levels. The researchers also indicated that it would be better if the panning and zooming interface in Eikon allowed smoother zooming of the data by automatically changing the number of glyphs or type of glyph used rather than having to do this manually. Other functionality, not included in Eikon but that was desired by the researchers included the ability to output images in postscript format suitable for inclusion in publications, the display of multiple small views in a single image or postscript document, the ability to generate animated sequences of different features overtime, and the application of the system to three-dimensional volume data. With the exception of processing volume data the other functional requirements can be readily added to Eikon. It may also be possible to extend the distribution process to volume data; this is discussed in the future work section in the Chapter 12.

Chapter 10

Case Study - Intermediate Views

In the global overviews presented in the previous chapter individual glyphs were too small to represent data through geometric variation and selected important characteristics of the data were highlighted through spatial distribution of glyphs within the image. More detailed information is obtained by interactively zooming areas of the image. With simpler and fewer glyphs interactive panning and zooming of the data is possible on a wide range of computer hardware. The distribution process allows the number of glyphs displayed to be dramatically reduced while preserving the most important features in the data. As a result more complex glyphs can be use while still maintaining interactive frame rates for data exploration. With fewer glyphs to be displayed each glyph can be rendered using more screen space thereby supporting more geometric variation. Together, the reduction in the number of glyphs with the ability to use larger glyphs, provides a method of producing intermediate level views that depict important features and allow other details to be displayed also.

There are many situations where intermediate level views are useful. For example, once a researcher has examined the global characteristics of a particular data variable, they may wish to determine the relationship of this data variable to other variables in the dataset to gain a deeper understanding of the data. Also when a global phenomenon has been understood focused intermediate level views can be used to communicate the important characteristics of it to others. This chapter explores the use of Eikon in the production of these intermediate views of the data.

Glyph Parameter	Minimum Value	Maximum Value	Data Variable
Bar 1	0.0	16.0	Height
Bar 2	0.0	16.0	Pressure
Bar 3	0.0	16.0	X Velocity
Bar 4	0.0	16.0	Y Velocity
Bar 5	0.0	16.0	Vorticity
Profile Size	0.0	10.0	Unbound
Color	0.0	0.0	Height

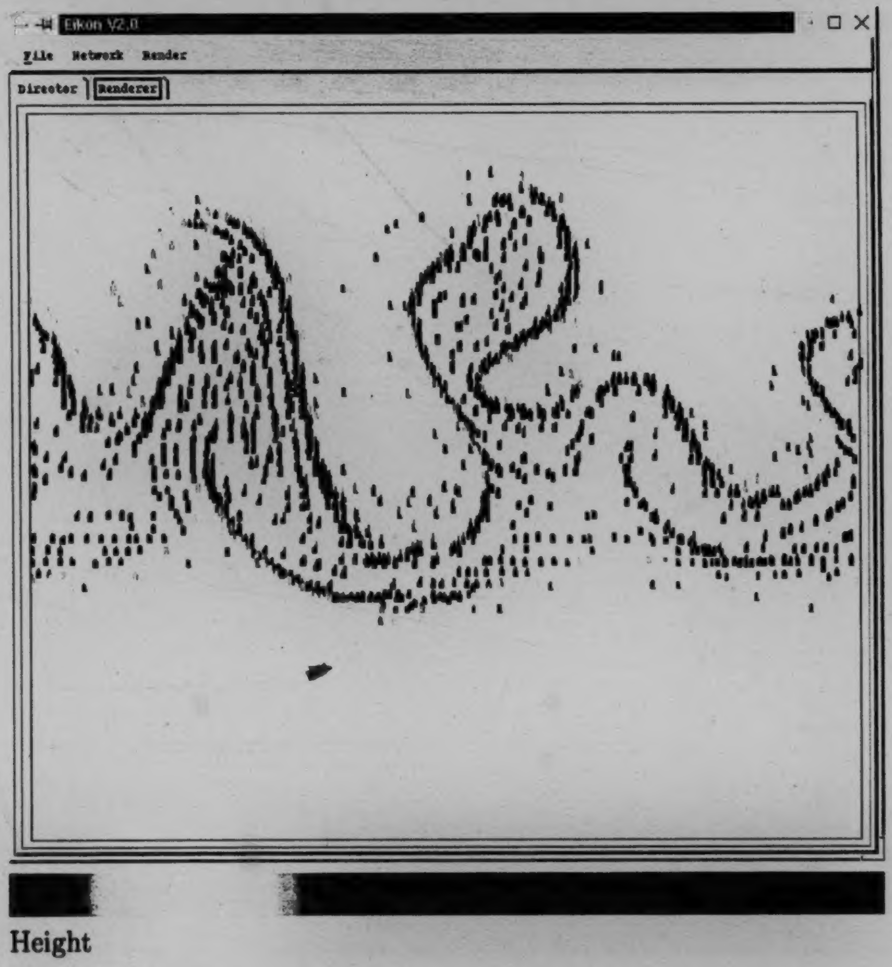
Table 10.1: The bindings assigned to the profile glyph (intermediate view).

10.1 Intermediate Views in Eikon

Figure 10.1 shows the GFD dataset rendered using 2,000 profile glyphs distributed using a variance importance function applied to the height data variable. Each glyph is displayed using the data-glyph binding in Table 10.1.

In Figure 10.1 not only is variation in height evident through the density and color of glyphs in the image but variation in some data points is also visible through the shape of individual profile glyphs. However, even at the increased size, this variation is small and borders on insufficient for showing details of data points. More importantly, however, as each data point is displayed at a size larger than the inter-data point spacing many of the profile glyphs overlap with neighboring glyphs adversely affecting the data variables they represent. This problem is similar to the glyph size and complexity problem discussed in Chapter 8 and can be rectified somewhat by zooming affected areas and changing the size of glyphs displayed. As fewer glyphs are used to render the image zooming can be accomplished at interactive rates. This problem can also be addressed by using less complex glyphs or glyphs less susceptible to interference from neighboring glyphs. Figure 10.2 shows another intermediate view of the GFD data using 10,000 stick figure glyphs distributed based on pressure magnitude and using the data-glyph binding described in Table 10.2

In Figure 10.2, like other images, pressure magnitude is evident through glyph density and velocity magnitude through glyph color. Unlike other images however some elementary and intermediate information is visible through individual and groups of glyphs. The orientation of the stick figure's main body is visible over a lot of the image and depicts variations in height over the dataset. Through this variation a

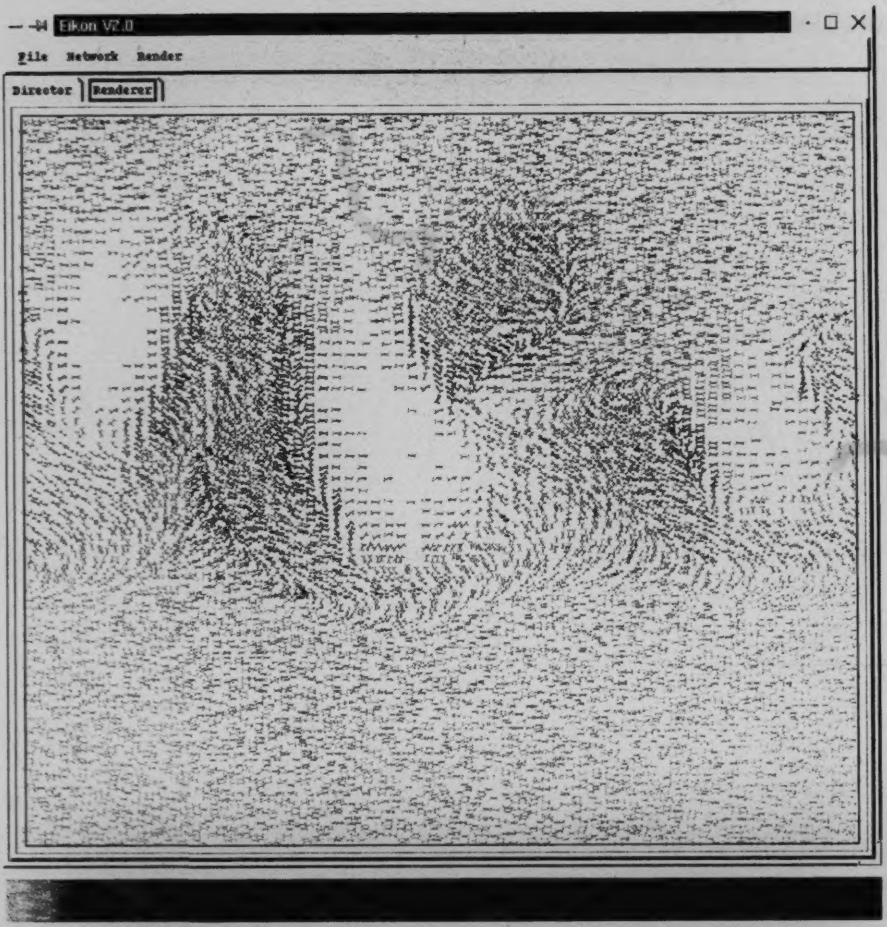


This image shows an intermediate level view focusing on variation in height, using 2,000 profile glyphs colored using the color sequence shown to also display height.

Figure 10.1: Height variation with a profile glyph (intermediate view).

Glyph Parameter	Minimum Value	Maximum Value	Data Variable
Limb 1 Orientation	0.0	180.0	Height
Limb 2 Orientation	0.0	45.0	Pressure
Limb 3 Orientation	0.0	45.0	X Velocity
Limb 4 Orientation	0.0	45.0	Y Velocity
Limb 5 Orientation	0.0	45.0	Y Vorticity
Limb Length	0.0	9.0	Unbound
Color	0.0	0.0	Velocity Magnitude

Table 10.2: The bindings assigned to the stick figure glyph (intermediate view).



This image shows an intermediate level view focusing on pressure magnitude, using 10,000 stick figure glyphs colored using the color sequence shown to display velocity magnitude.

Figure 10.2: Pressure magnitude using the stick figure glyph and color.

Glyph Parameter	Scale	Data Variable
X Coordinate of End Point	0.15	X Velocity
Y Coordinate of End Point	0.15	Y Velocity
Z Coordinate of End Point	1.0	Unbound

Glyph Parameter	Minimum Value	Maximum Value	Data Variable
Size of Bulb	0.0	3.0	Pressure
Color	0.0	1.0	Vorticity

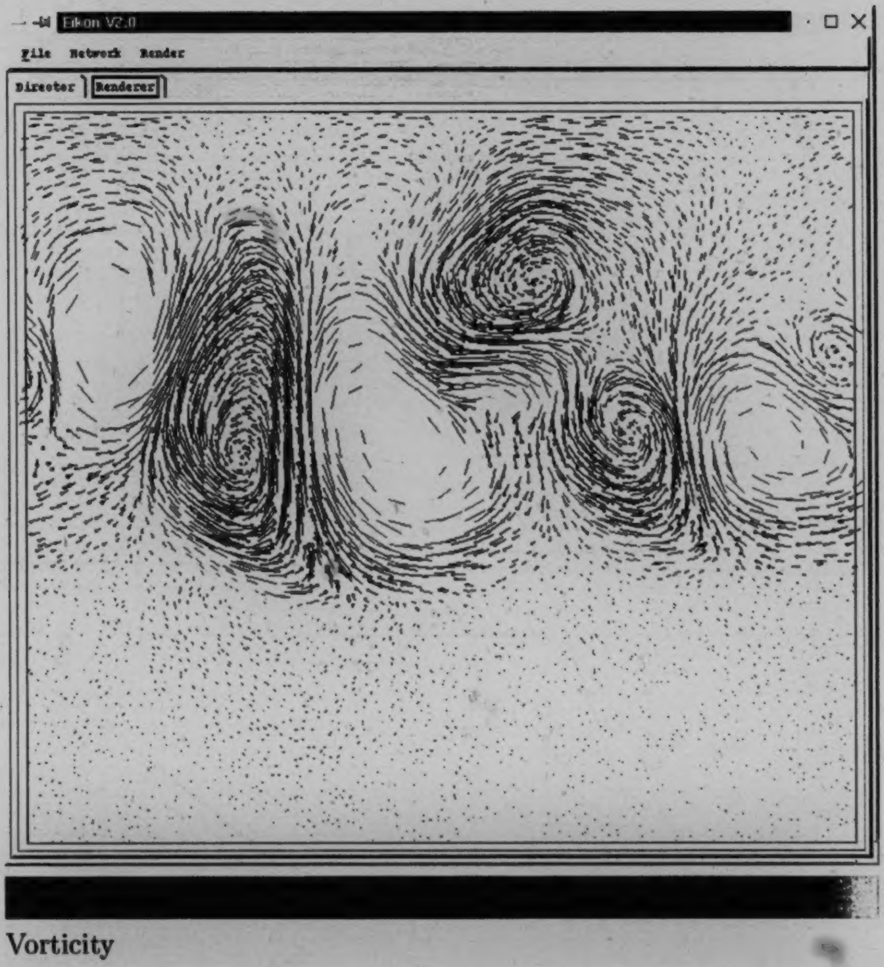
Table 10.3: The binding assigned to the line glyph (intermediate view.

texture is evident reflecting areas where height is uniform and where it changes. The texture regions are not as clear as those in Figure 8.7 however. In Figure 8.7 the regular placement of glyphs and the larger size of individual glyphs resulted in distinct texture in the image. In Figure 10.2 the non-uniform placement of glyphs disrupts this visual texture somewhat and the overall size of each glyph reduces visible variation of limbs other than the main body. Also in dense areas the variation in geometry needed to create the texture is masked.

Figure 10.3 shows another view using 3,000 line glyphs distributed based on pressure magnitude. This image uses the binding in Table 10.3 and in this case the reduced number of glyphs displayed, the simplicity of the line glyph, and the direct correspondence between line orientation and magnitude to flow parameters produces an image where details of the flow are visible over the entire image. The density of glyphs clearly indicates where pressure is large and small, the size of the bulb on each glyph depicts height and finally vorticity is illustrated through the color of each glyph.

10.2 Importance Functions and Multiple Variables

The amount of information depicted in an intermediate view can be increased by applying the importance functions to a combination of data variables or abstractions. In this way images can be generated that depict areas where, for example, pressure *or* height are large. Using the various abstraction, variable extraction, and merging tools, any combination of data variables or abstraction can be provided to an importance function. Figure 10.4 shows the output of applying the magnitude importance



This image shows an intermediate level view focusing on pressure magnitude using 3,000 line glyphs colored using the color sequence shown to display vorticity and the size of the bulb to show height.

Figure 10.3: Pressure magnitude using line glyph and color (intermediate view).

Glyph Parameter	Scale	Data Variable
X Coordinate of End Point	0.15	X Velocity
Y Coordinate of End Point	0.15	Y Velocity
Z Coordinate of End Point	1.0	Unbound

Glyph Parameter	Minimum Value	Maximum Value	Data Variable
Size of Bulb	0.0	-3.0	Pressure
Color	0.0	1.0	Pressure/Vorticity

Table 10.4: The binding assigned to line glyph (height and pressure view).

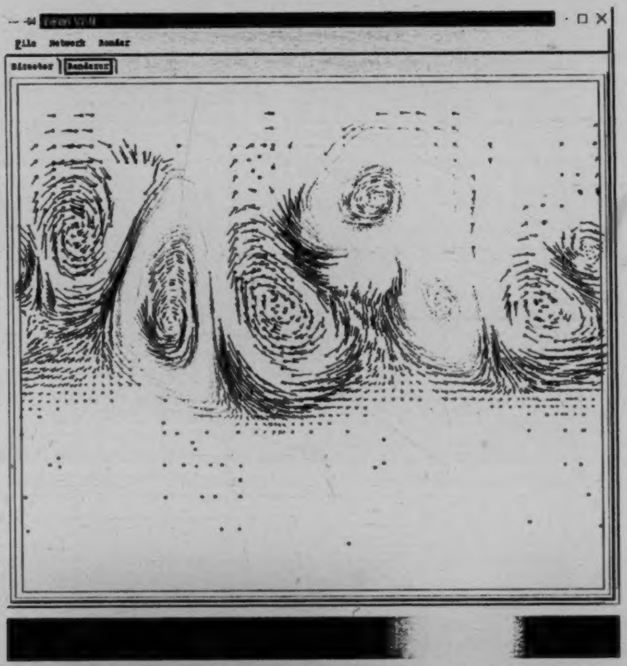
Glyph Parameter	Scale	Data Variable
X Coordinate of End Point	0.15	X Velocity
Y Coordinate of End Point	0.15	Y Velocity
Z Coordinate of End Point	1.0	Unbound

Glyph Parameter	Minimum Value	Maximum Value	Data Variable
Size of Bulb	0.0	-3.0	Pressure
Color	0.0	1.0	Height/Vorticity

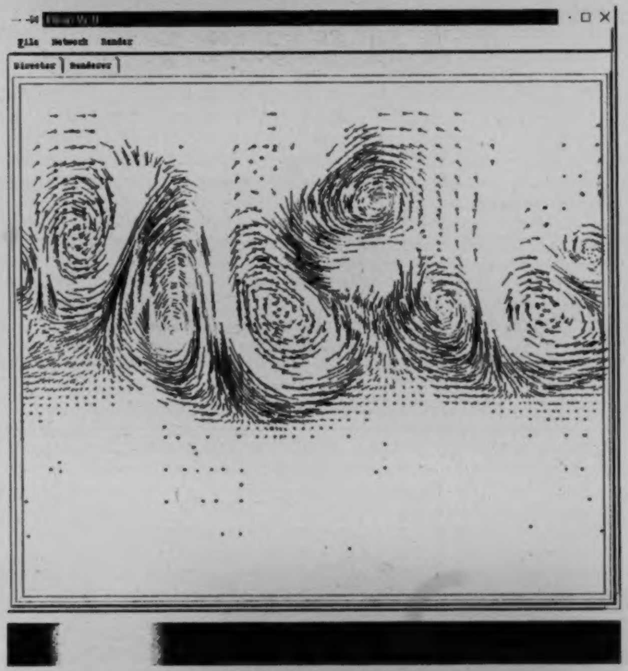
Table 10.5: The binding assigned to the line glyph (pressure and vorticity view).

function to generate an image showing magnitude of pressure and height using 4,000 line glyphs and the binding in Table 10.4. Figure 10.4(a) colors each glyph based on the height while Figure 10.4(b) colors each glyph based on vorticity. Similarly Figure 10.5 shows the affect of using the magnitude importance function with pressure and vorticity. This image also uses 4,000 line glyphs with the binding in Table 10.5. Figure 10.5(a) uses height to color each glyph and Figure 10.5(b) uses vorticity.

The effect of using multiple variables is that it allows glyphs to be placed in other locations based on the composition of the selected variables. This enables details that would not have been displayed using one variable to be seen. However it also has the effect of making it ambiguous as to which feature of the composition is displayed by dense regions in the image. If it is important to discriminate between the features they could be rendered separately at first to determine their locations and then in a combined image, or, another visual variable could be use to distinguish regions in the image. For example in Figure 10.4(a) the size of the bulb indicates areas of high pressure and low pressure while color also represents areas of large and small height values.



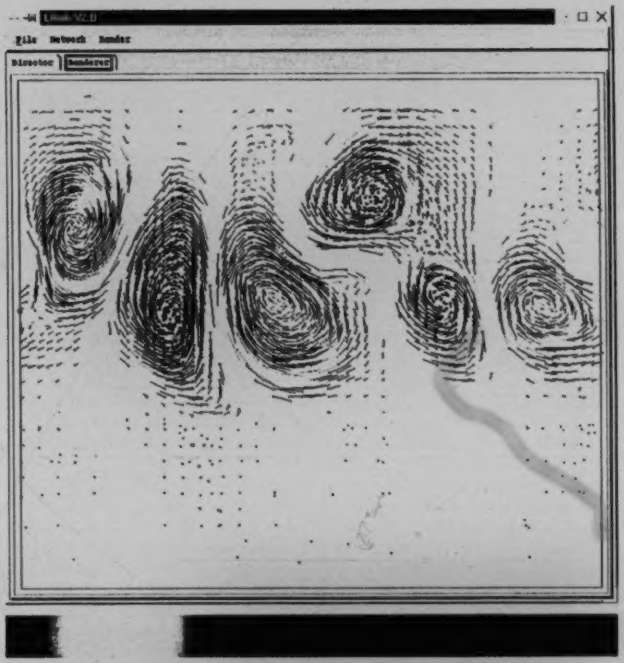
(a) Height and Pressure colored with Pressure



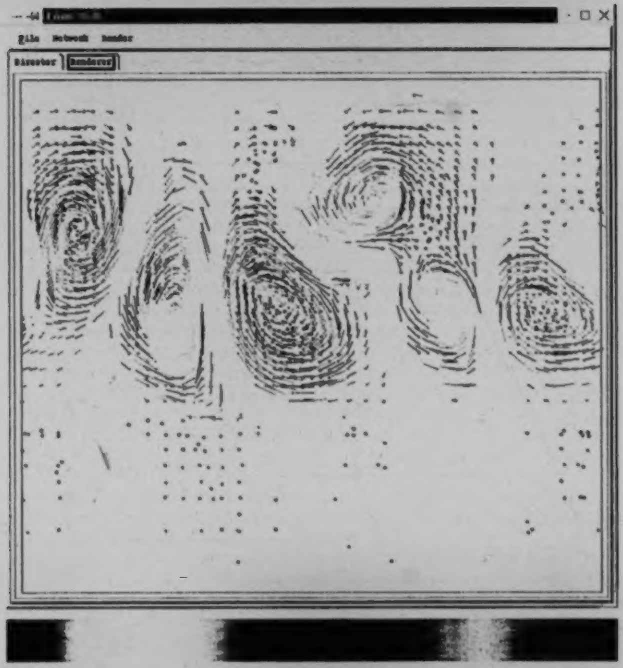
(b) Height and Pressure colored with Vorticity

The GFD data rendered using an importance function applied to two data variables.

Figure 10.4: Pressure and height combined.



(a) Pressure and Vorticity colored with Height



(b) Pressure and Vorticity colored with Vorticity

Figure 10.5: Pressure and vorticity combined.

In several of the previous images it was possible to determine the direction and magnitude of the flow based on the glyphs drawn. However in some cases the density of glyphs obscured this variation in individual line glyphs. These views contain only a few thousand glyphs and therefore can be panned and zoomed interactively. This allows interesting areas to be quickly explored by zooming, providing more detail for individual glyphs. Figure 10.6 presents several examples of zoomed regions in the GFD dataset that correspond to smaller (but still large) scale intermediate views. This views display around one half of the data points in the dataset thereby affording even more space for individual glyphs. As a result more geometric variation is visible, providing increased detail for individual data points, while density and color provide a broader overview of other characteristics in the displayed data.

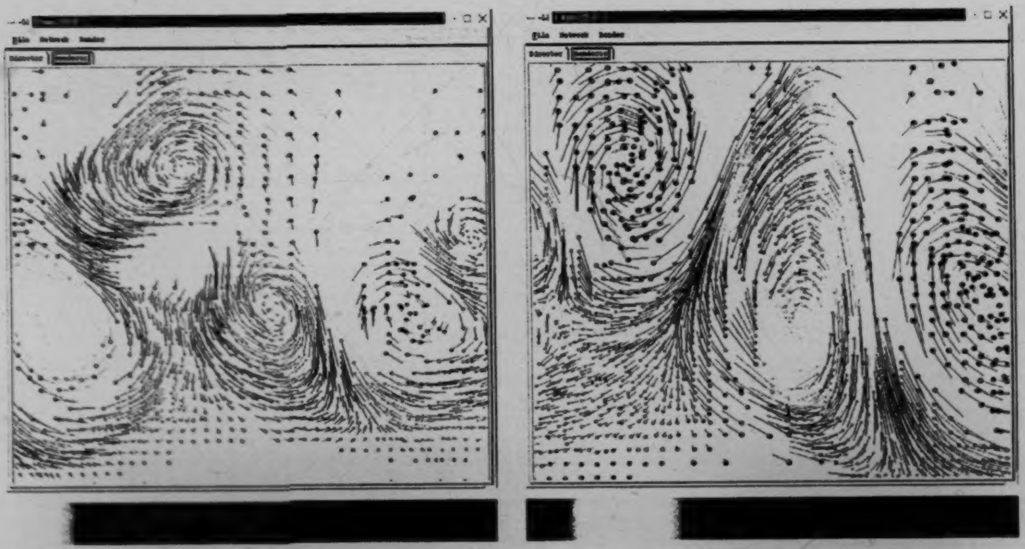
10.3 Multi-glyph and Multilayer Views

Beyond using different visual variables from a single glyph to make distinctions between different data characteristics, Eikon also allows multiple glyphs to be used. In Chapter 8, I showed how multi-glyph techniques can be implemented within Eikon by rendering an extract from the data with two or more glyphs and where the visual attributes of the glyphs were adjusted to either increase or decrease their visual impact.

Within Eikon this method can be further augmented by using an importance function to govern the distribution of the glyphs within the image. Figure 10.7 shows an example image of the data rendered using the line and profile glyphs distributed over the image using a magnitude importance function applied to the height data variable. The line glyph is colored using the pressure variable as an index into the color scale shown in Figure 10.7. The profile glyph is colored using pressure as an index into a gray scale. The binding for the profile glyph is described in Table 10.6

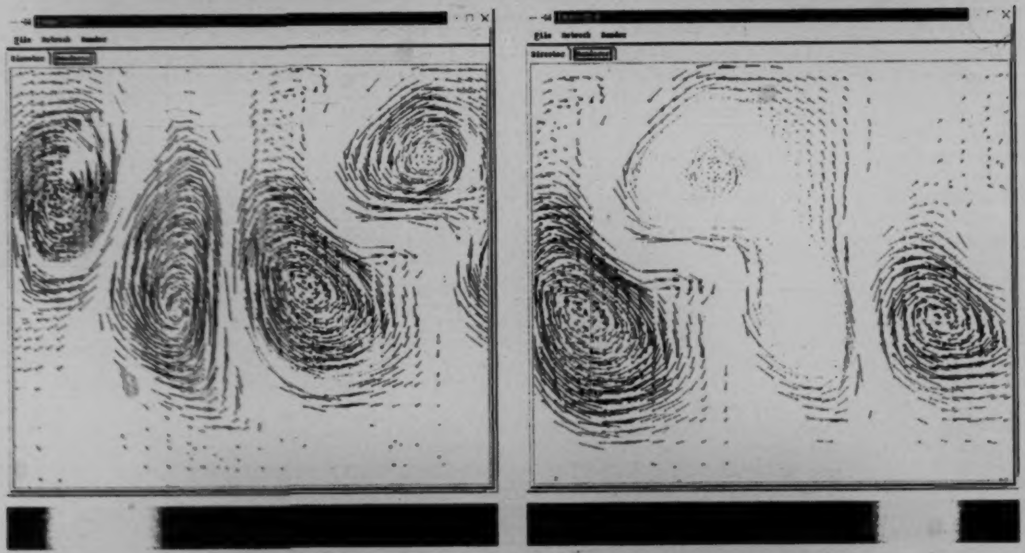
10.3.1 Multilayer Views

Figure 10.8 shows an example of a multilayered view based on two extracted regions of the data. This view is similar to Figure 8.12 in Chapter 8 with the addition that the glyphs within each extracted region are displayed using different glyph distributions.



(a) Pressure, Height, Pressure

(b) Pressure,Height, Vorticity

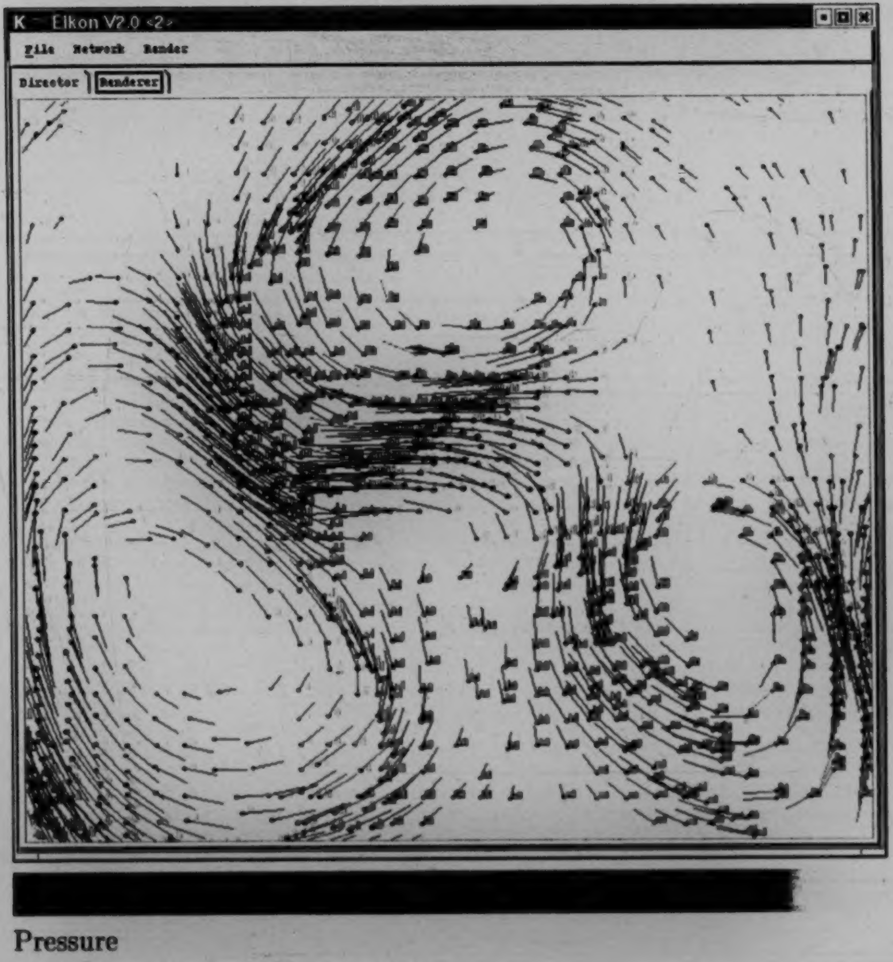


(c) Pressure, Vorticity,Height

(d) Pressure, Vorticity, Pressure

The data rendered at a zoomed level using importance applied to a combination of data variables and using different colors.

Figure 10.6: Zoomed intermediate views.



This image shows an intermediate level view focusing on magnitude of the height variable. It uses two glyphs, the line and the profile glyph.

Figure 10.7: Multiple glyphs using a single importance function.

Glyph Parameter	Minimum Value	Maximum Value	Data Variable
Bar 1	0.0	16.0	Height
Bar 2	0.0	16.0	Pressure
Bar 3	0.0	16.0	X Velocity
Bar 4	0.0	16.0	Y Velocity
Bar 5	0.0	16.0	Velocity Magnitude
Profile Size	0.0	10.0	Unbound
Color	0.0	0.0	Pressure

Table 10.6: The bindings assigned to the profile glyph.

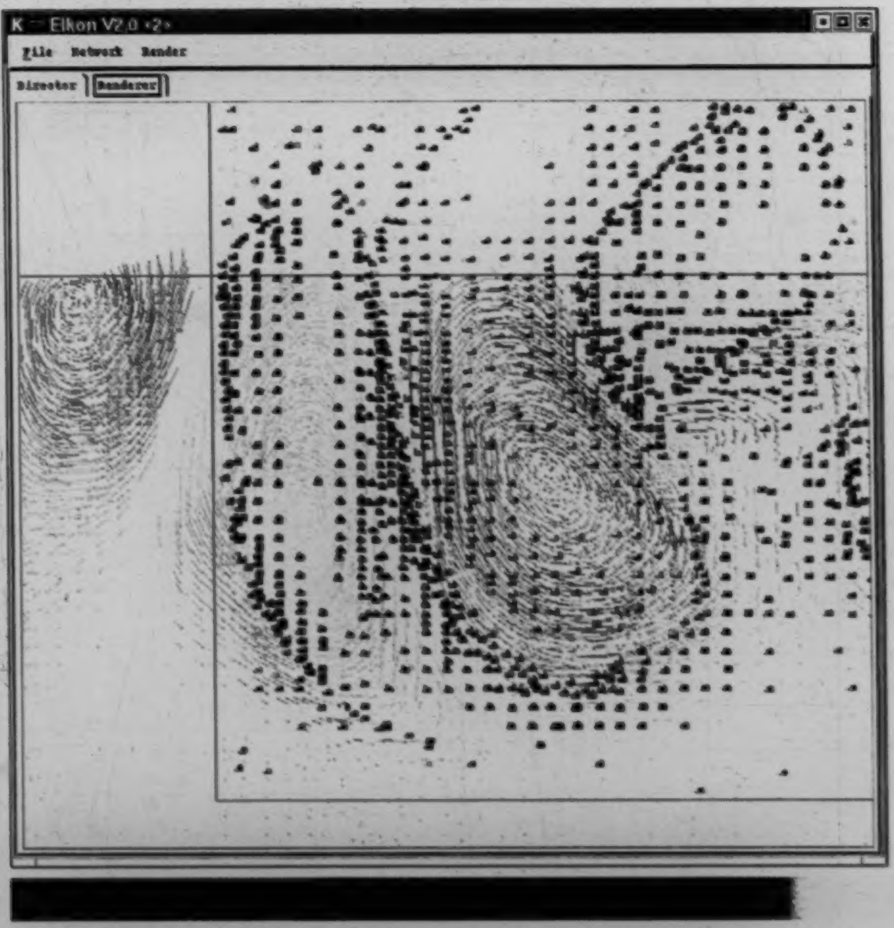
In this view line glyphs are distributed using an abstraction based on vorticity and using a magnitude importance function. The profile glyphs are distributed based on the height data variable also using a magnitude importance function. The line glyphs are colored using a gray scale based on the computed vorticity value while the profile glyphs are colored using the color scale shown in Figure 10.8 and the bindings described in Table 10.6. In this view one layer allows us to see patterns and structures associated with vorticity and velocity of the data, while the other allows us to see patterns and structures associated with height and pressure variations. Where the views overlap we can compare and contrast the features associated with vorticity and velocity with those of height and pressure. The views also display the data with a reduced number of glyphs and support interactive panning and zooming of the data.

10.4 Discussion

In this section I showed how detail at a global level can be traded for more detail on a per data point basis. In this way views that focus on the most salient features of the data (selected by the user) and that display other details of the data can be constructed. These images contain fewer glyphs allowing each glyph to be drawn at an increased size and exhibit more geometric variation. As such views generally contain only a few thousand geometric primitives supporting interactive panning and zooming of the data displayed using a wide variety of glyphs on a wide range of computer hardware.

10.4.1 Advantages

In the context of the case study, the researchers often like to see as much detail as possible while at other times it is necessary to focus attention on particular aspects of the data. Typically upon initial exploration it is sufficient to look at broad overviews of the data without being encumbered with unnecessary details. These types of views are provided through glyph density in Eikon as previously described. The researchers however also noted that as the understanding of the data increases it is usually beneficial to view more details of the data. This allows the researcher to determine relationships between different variables in the data and see precisely



Two regions are extracted and rendered using two different glyphs and two different distribution methods. One importance function is applied to the height data variable and one is applied to a data abstraction computing vorticity.

Figure 10.8: Multi-layered, multi-glyph image using two different distributions of the profile and line glyph.

what is going on. In terms of the GFD mathematical model this gives a deeper understanding of the data and allows the correctness of the model to be verified. Conversely, once a deeper understanding of the data is obtained it is often necessary to reduce the amount of detail, focusing only on data that is necessary to explain the phenomena. In this respect control over detail enables the construction of images suitable for communication of the research results to others.

In general the ability to adjust the amount of detail displayed in an image as desired using glyphs, importance, and abstraction combined as needed, was viewed as a major benefit. The visual programming environment allows the number of glyphs displayed and the data-glyph binding to be easily adjusted. Combined with the ability to change importance and abstraction functions the researchers agreed that this provided the flexibility needed to render the data with the amount of detail desired.

Although the researchers agreed that adding more detail can indicate relationships between variables that may not have been expected and thereby provide new insight into the data, adding more data can also cause some problems. Within Eikon concerns where raised about glyph density obscuring potentially meaningful geometric variation in interesting areas. However the researchers agreed that the ability to reduce the number of glyphs, with the ability to zoom the data and change the importance function if necessary was a reasonable solution to this problem.

10.4.2 Disadvantages

Similar to the concerns conveyed in Chapter 8 the researchers reiterated the concern relating to understanding the data encoding used by particular glyphs. The researchers felt that some glyphs (such as the face glyph) required a lot of effort to determine which characteristics of the glyph geometry were producing the patterns in the image and what the relationship between glyph geometry and data variable characteristics were. The researchers referred to this as information overload as they could not easily extract the information needed from such glyphs. Throughout the case study the glyph preferred most by the researchers was the line glyph. This was primarily due to the direct relationship between glyph geometry and flow characteristics, the simplicity of the glyph itself, and to its already common use in the field.

In fact the researchers were hesitant to go beyond images that displayed any more information than was possible using line glyphs, density, and color. The research conducted in the development of Eikon was not concerned with the properties of individual glyphs but the extension of geometric coding in general to large multivariate datasets. The definition of other effective geometric codes is beyond the scope of this thesis and remains as an open research problem.

Finally the researchers again indicated that the abstraction, importance, and distribution method in Eikon might be usefully combined with other visualization techniques such as color images and contour plots. They also suggested that for precise elementary information that the user have the ability to point at a data point and have its exact data values displayed.

Chapter 11

Case Study - Conclusion

My objective with this research is to show that Eikon, a system for large multivariate data visualization, based on the concepts of using abstraction, importance, and distribution, is a useful system for extending geometric coding to the analysis and exploration of large multivariate datasets. To show this, in the last 4 chapters, I described the results of using Eikon in a case study conducted with researchers in the fluid dynamics research group at the University of Alberta. In this chapter I summarize the benefits and shortcomings of Eikon from these results and draw some conclusions with respect to its utility.

11.1 Benefits

The results of the case study show that the abstraction, importance, and distribution process, and Eikon itself, are useful for applying geometric coding to large multivariate datasets. As indicated in Chapter 2, a useful image is an image that meets the requirements of the user and by definition the most useful image is one in which all user questions pertaining to the data can be answered. In general it is impossible to construct a single image that can represent the answers to all possible questions and as a result it is necessary to construct images in which the answers to preferred questions are easily visible. Questions can also be asked at a number of different scales, or levels of reading, in which varying amounts of detail about the data are required. The abstraction, importance, and distribution process is useful as it provides a mechanism by which the amount of information displayed can be adjusted based on the user's needs. This allows more effective use of the limited visual variables to display data that allows users to determine the answers to various questions at various levels of reading.

11.1.1 Global Overviews

Global overviews display all the data points in a data set but display little detail with respect to individual data points. The objective of these views is to focus on particular features in the data at a global scale. Such views are particularly useful when the data is first encountered as typically the user interest lies in the global structure and patterns of important features at this time. These views are generated in Eikon by using the abstraction, importance, and distribution process to distribute numbers of glyphs over the image generating images where important features are evident through the density of glyphs in the image. Other features of the data can also be represented through color. These views were useful as they allowed the researcher to clearly see the patterns and structures of important features without being distracted by unnecessary details. Also the ability to define and use new abstraction and importance functions aided the visual exploration and understanding of the data. Using color for individual glyphs allowed relationships between the feature display through glyph density and other features in the data to be determined. The color sequence editor in Eikon also provided an adequate method for specifying color sequences to use. Finally although many glyphs may be needed to display the details of the global patterns and structures, at this scale, the individual geometry of each glyph contributes little to the display thereby allowing simple square glyphs to be used. This allows interesting areas in the data to be interactively explored through panning and zooming. On hardware that has problems displaying simple glyphs the number of glyphs could also be reduced while maintaining the important features of the patterns and structures displayed.

11.1.2 Intermediate Views

Intermediate views display large groups of data points and allow the user to determine patterns and structures over local neighborhoods of the data. These views are similar to global overviews however global detail is traded for increased detail on a per data point basis. In the case study these views where used to explore the data after the location of global features of interest where determined. In this case the focus of the researcher moves away from the precise structure of the global features towards more local details associated with the features. In Eikon the increased detail in intermediate views is achieved by reducing the number of glyphs used to display the global feature, enabling larger glyphs to be used and more geometric variation to be visible. Again the ability to change the number of glyphs displayed, the type of glyph used, the binding between glyph parameters and data variables, along with the use of abstraction, importance, and distribution, to place glyphs and the ability to specify different color sequences provided the flexibility and control needed to adjust the focus and detail presented in the image as necessary. As the intermediate level views typically contained only a few thousand glyphs interactive panning and zooming is supported using more complex glyphs and allows the user to smoothly move from intermediate level views to more elementary level views and back quickly and easily.

11.1.3 Elementary Views

Elementary views display a small focused set of data points and display all detail associated with their data variables. Such views support the most basic level of reading and are useful for recovering the precise nature of the data at select data points or groups of data points. Eikon generates elementary views of the data by simply placing a data glyph at each data point and does not use the distribution or importance mechanisms. Abstractions can be used to compute new data that can be represented by parameters of individual glyphs. Also the color of individual glyphs can be used to represent other data variables. Eikon also allowed different glyphs to be easily used to represent different elementary views and for the binding between data variables and glyph parameters (including color) to be readily changed. This facilitates the use of different glyphs and bindings for different visualization purposes at the elementary level.

11.2 Shortcomings

Eikon is not a *deux ex machina*[1] and several problems with the processes on which it is based and with the system itself were identified.

11.2.1 Discrete Displays

The main disadvantage noted with the use of glyph distribution was the discrete nature of the images produced. Glyph based displays in general are discrete typically only placing glyphs at defined data points. This is compounded by the distribution process which selectively places glyphs at different (typically non-uniform) locations in the data. Even though simulations and direct measurement generate discrete data points, these are representative of a continuous phenomenon. As a result the viewer may wish to see a visual representation that reflects this continuity. Using the distribution process areas of the data may not be displayed which produces holes in the data and is not representative of the continuous nature of the data. The perceived lack of continuity was addressed in part by associating an outline with the area of the data displayed. This outline served to distinguish the significance of the space it contained, as being part of a continually varying phenomenon, from the less significant space outside, which might also be part of a continually varying phenomenon but is unimportant in the current display. In the context of the case study the researchers also suggested that glyphs be displayed on top of a continually varying color image. The color image indicates the underlying continuity of the data and allows other data variables to be represented through color variation.

11.2.2 Color Variation

Color variation relates to another problem associated with using geometric coding and glyph distribution to display information. Color variation can affect the visibility of individual glyphs and can change the perceived density of glyphs in the image. This is the case when individual glyphs are colored and would also be evident if glyphs where placed on a continuously varying color image. Although the color sequence editor provides a mechanism for defining color sequences no guidance is provided in constructing appropriate ones. A simple solution to this problem would support a number of predefined color sequences with particular properties or to include an "assistant" to guide the user in the construction of color sequences.

---

[1] Or more appropriately it is not a *theos ek mekhanes*, to quote the original Greek form

11.2.3 Local Detail

Considering more intermediate and elementary level views, problems with glyph density and visible detail were also noted. Using the distribution process density of glyphs reflects the importance of a data location. However increased glyph density reduces the amount of space for geometric variation and obscures some of the detail displayed. This problem can be partially solved by reducing the number of glyphs displayed such that enough space is available, by applying some maximum ceiling to the number of glyphs for any one region, or by otherwise effecting the importance function such that a good distribution is obtained. The ability to quickly zoom interesting areas also reduces the affect of the density problem and it may be possible to apply other scale-space methods to allocate more space to these important areas.

Another problem raised dealt with the complexity and legibility of different glyphs. From the case study complex glyphs such as the face glyph were not held with much regard. With complex glyphs it was difficult to see variation when small glyphs were displayed and even when enough resolution was available it was difficult to determine the relationship between visible variation in the image and the corresponding data variation that caused it. The line glyph was the most favored glyph because of the direct relationship between glyph and data variation. The effectiveness of different glyphs was not regarded as a problem in the context of this work as Eikon allows different glyphs to be used easily and when global overviews are rendered individual glyph geometry is less important. None-the-less the design and evaluation of effective geometric codes is an interesting research problem.

11.2.4 Other methods

The final shortcoming relates to the lack of other more standard tools in the Eikon system itself. Researchers desired other tools such as color images, contour plots, animations, postscript output, multiple small views etc. The flexible design of ETK on which Eikon is based and the architecture of the visual programming environment should allow extra functionality to be added without much trouble. Also the structure of ETK resembles the popular and freely available Visualization Toolkit, suggesting it could be combined with it with some rework. This would provide access to a wide variety of more traditional visualization methodologies.

11.3 Conclusion

Overall, Eikon and the abstraction, importance, and distribution method it contains proved a valuable way of using geometric coding methods in the visual exploration and analysis of large multivariate datasets. Despite the noted deficiencies, the system and methodology provide a platform by which the amount of information encoded by the visual variables of an image can be controlled based on user requirements. The system allows different abstraction functions, importance functions, and glyphs to produce multiple different views of the data. Global overviews can be generated based on glyph density and color, depicting important features independently of specific geometric properties of glyphs. Various other level of detail views can be generated by reducing the number of glyphs displayed and using glyphs with appropriate geometric properties. In these views important features are visible via the density of glyphs while other details are visible through glyph color and geometric variation. Other intermediate and elementary views can be generated by simply ignoring the distribution process and placing a glyph at each data point in a manner similar to many existing geometric coding methods. The utility of Eikon is further improved by supporting this functionality within a single system that allows visualizations to be constructed and configured quickly using a visual programming environment. This allows the various parameters that affect the image created to be quickly and easily changed allowing the user to visually explore the data more easily.

Perhaps the most important aspect, with respect to the usefulness of Eikon, is that as a result of the case study there is now interest in further application of the system and extension of its capabilities to meet some of its shortcomings.

Chapter 12

Conclusion

In this thesis I presented a system that extends the capabilities of geometric coding methods to large multivariate datasets. Geometric coding allows many data variables to be represented in a single image using parameterized geometric objects or glyphs. By parameterizing the glyph properties with respect to data variables, values and relationships of individual or small groups of data points can be seen directly from glyphs in the image. Furthermore, when appropriately designed glyphs are displayed together, intermediate and global patterns and structures in the data can be evident through the visual properties generated.

The increase in the number of data variables that can be encoded using geometric methods is achieved at a cost of spatial bandwidth: given an image of fixed size, as more data points are displayed the space for individual glyphs is reduced. As a result, the amount of visual variation supported by each glyph is reduced; this can generate images that obscure important structures and patterns. In addition the large number of graphic primitives that must be displayed can place undue strain on the computation and display hardware.

The work I presented here extends the capabilities of geometric coding by capitalizing on two observations: (1) the analysis of large multivariate datasets is not always facilitated by the display of all data points and (2) the spatial dimension is often underutilized in current geometric coding methods. I exploit these observations by making more effective use of spatial dimensions and selectively displaying only important data. When the ratio of pixels to data points is low, this scheme provides overviews of the data through the variation of glyph density in the image. When the ratio of pixels to data points is high, variation in individual glyphs becomes more effective; however, variation in glyphs density can still be used to highlight structures in the data.

The system that I presented in this thesis supports such displays by using *abstraction functions* as a way of computing essential or interesting features from the data, *importance functions* to determine important characteristics of the abstraction, and the *multi-scale distribution* of glyphs over the image using the computed importance to display the data. I also presented a visual programming environment which allows visualizations to be constructed interactively and that allows the parameters associated with the visualization to be easily changed, allowing the user to visually explore the data they are interested in.

12.1 Abstraction and Importance

Abstraction is used to extract information that in some way is interesting to the user. Abstraction is realized as a user-definable function that is applied to the dataset and that returns a set of values indicating the output of the abstraction function at each point. Abstractions perform the first step in the display process and can be conceptualized as a method of querying the data where the output of the query forms the basis of how the data will be rendered.

Importance functions are applied to the abstractions (or original data) to determine the spatial distribution of glyphs in the image. These functions determine the relative significance of each data point by determining the degree that each data point exhibits a particular characteristic associated with the importance function. For example, a variance importance function computes the amount of data variation within the neighborhood of each data point giving high significance to data points in areas of high variation.

The spatial distribution of glyphs is controlled by computing a multi-scale mipmap type representation of the importance values and using this to distribute glyphs to different regions in the image. The multi-scale importance pyramid is represented by a quad-tree structure where the leaves in the tree correspond to the computed importance values. Higher level nodes in the quad-tree are computed by averaging the associated four nodes at lower levels until a root node (or apex) is computed representing the average of all importance values in the data.

12.2 Distribution and Data Display

Distribution begins by assigning the number of glyphs to be distributed to the apex of the pyramid and distributing glyphs to the four regions at the next lowest level. The number of glyphs distributed to each region is based on the region's importance value relative to the value of the other three neighboring regions. The distribution process then recursively distributes glyphs to the four sub-regions at the next lower level in the pyramid associated with each of the current regions and so on until all glyphs have been distributed. The resulting glyph distribution is an image in which the important features of the abstraction are visible through the density of glyphs. Patterns and structures are visible through the size, shape, orientation, and intensity of the features created through variation in glyph density.

12.3 Benefits of the current system

When glyphs are used to display the data it is important to be able to read the data at global, intermediate, and elementary levels. The scale problem associated with glyphs reduces the legibility of images at the global and intermediate levels. This thesis extends the capabilities of geometric coding for both global overviews and intermediate views. The primary concern is not the design of geometric glyphs themselves, so little is contributed at the elementary level of reading.

12.3.1 Global Overviews

Global overviews of large datasets can involve displaying data with only a few pixels available for each data point. At an extreme one or less pixels may be available for each data point, and variation in individual glyphs at this point is not effective. The developed system enables the use of geometric coding at such scales in the following ways:

- Displaying large numbers of glyphs can place a large rendering load on the computer and place little emphasis on any of the data. The use of abstraction, importance and glyph distribution means that views can be generated which focus on various important aspects of the data by distributing a user definable number of glyphs over the image based on the location of the identified features.. This process allows the number of glyphs to be dramatically reduced but at the same time ensures that most important features remain visible through glyph density in the image. Changing the abstraction and the importance function allow different images which focus on different data characteristics to be displayed.

- When little space is available to render each glyph, variation in the density of glyphs is used to represent important characteristics of the data. Important patterns and structures are then visible through the position, size, shape, and orientation of the features generated in the image and the color of individual glyphs can be used to depict other information. Depicting information through variation in density and color allows very simple glyphs to be displayed which when combined with the ability to reduce the total number of glyphs displayed allows the data to be interactively explored.

12.3.2 Intermediate and Elementary Views

From the global view the user can identify important areas using variation in density of glyphs and these areas can be explored more closely by zooming at which point individual glyphs will occupy more space and support more visual variation. This provides a seamless way of moving from global overview to an intermediate and elementary view without loosing context or adjusting how the data is displayed. The system contributes to intermediate level views in the following ways:

- For elementary or small scale intermediate views, involving only a few thousand data points, traditional geometric coding methods can be emulated by simply placing a glyph at each data point in the image.

- The current technique is less dependent on the geometric properties of glyphs. The exact nature of the non-local features created by uniformly placing glyphs over the display is dependent on the geometric properties of each glyph, the data-glyph binding used, and the size in screen coordinates of the rendered glyph. By basing the display on glyph density non-local features can be rendered using density irrespective of the geometry of individual glyphs.

- With appropriate glyphs, larger scale intermediate views, involving tens, or hundreds, of thousands of data points can be generated. In these views the number of glyphs used to depict global information can be reduced allowing more space for individual glyphs. This allows individual glyphs to display more local data variation. As a result the system can generate enhanced intermediate views which depict features of interest through glyph density and color, and which depict more local detail through geometric variation. Again the ability to control the number of glyphs displayed generates views which can be interactively explored.

12.3.3 Visual Programming Environment

The ability to construct networks using multiple abstractions, importance functions, and data glyphs is supported using Eikon: a system that couples an integrated visual programming environment with the data readers, filters, and writers, along with, abstraction functions, importance functions of the underlying ETK. This programming environment allows visualization networks to be constructed visually, as collections of processes from ETK, and allows the output of these networks to be displayed and interacted with. The programming environment also allows the individual parameters governing the behavior of the visualization to be changed and allows the results of these changes to be seen. This provides a useful environment for producing multiple different views of the data based on different abstractions, using different importance functions, different data glyphs, different numbers of data glyphs, and different data-glyph bindings. This supports visual exploration and analysis of the data and provides the necessary control over how much information (detail) is displayed and how the visual variables can be effectively used to represent it depending on the needs of the user.

The environment also allows the created visualizations to be exported as C++

code and run as stand alone programs which can be further tailored to meet user requirements. The ability to access the source code of ETK and the visual programming environment, coupled with the ability to extend both with new processes and tools, along with access to source code for created visualizations, creates an effective visualization development environment that can meet future visualization needs.

12.4 Limitations and Future Considerations

Although two-dimensional uniform structured grids are an important class of data grid for scientific computing, other non-uniform grids are also common. The current system assumes that data points are uniformly dense throughout the dataset and does not account for the non-uniformity present in other grids. For example, curvilinear grids normally have denser grids near interesting areas and have other non-uniformities due to the curved nature of grid cells. Such grids can be used in Eikon, but they must first be re-sampled onto a uniform structured grid. A better solution to this problem takes into account the shape and area defined by grid cells, and uses this to adjust the weighting assigned to different areas.

The methods in Eikon have only been applied to two-dimensional data grids. The implementation does support the application of the abstraction, importance, and display method to three-dimensional data grids; however more work is needed to determine the usefulness of such methods. The distribution method may not work well in three-dimensional space as it may be difficult to determine the structures from the distributions of glyphs. Many of the cues used to determine three-dimensional structure, for example occlusion, would be missing and may only confuse the viewer.

One area of application with respect to three-dimensional data grids that seems more useful is the distribution of glyphs over an iso-surface. In a manner similar to the dividing cubes algorithm [clin88], which recursively divides space until an iso-surface is encountered, glyphs could be distributed throughout three-dimensional space so that important areas of the iso-surface are displayed in more detail than unimportant areas. Initial experiments have shown that the iso-surface is rendered as a point cloud of glyphs which allows some structure to be determined. More work is required.

Eikon is limited to data in which two continuous dimensions are defined: there must be two dimensions that can be reasonably associated with the spatial dimensions of an image. Many applications in the scientific computing domain fall into this bracket as the data they contain is already defined in three-dimensional space. In the more general field known as "information visualization," it is often not clear how spatial dimensions should be employed. Therefore an interesting extension to Eikon would be the application of the abstraction/importance method to non-spatial data.

The use of abstraction and importance provides control over relatively large-scale features in the generated image. Finer control of visible features is more dependent on the geometric properties of the glyphs used. A future development of Eikon would involve extending control to finer details in the image. One approach would assign different types of glyphs to different levels of importance, or use a level of detail approach for glyphs so that when glyphs are small one representation is used but when glyphs are larger a more detailed representation is used. Individual glyphs themselves can be looked at as mini-images and knowledge from perception, in combination with importance values could be used either to automatically generate appropriate glyphs or suggest a number of possible choices. This work would also involve the design and evaluation of geometric codes for particular visualization tasks.

In its current state, Eikon supports only the manipulation of the various parameters for processes in the visualization network from within the visual programming environment. Once code has been exported from this environment, the user interface components associated with each process are no longer accessible. Modifying parameters of the visualization at this point involves changing the source code and recompiling the visualization. A useful extension would be to allow the exported code to access the various data fields and configuration dialogs that are available in the visual programming environment so that visualization parameters can be adjusted at run-time.

The ability to modify the display of one view and have this modification propagate to other views of the data has been identified as an important tool in data analysis and exploration [buja91]. Although Eikon is capable of generating multiple views that focus on different aspects of the data, there is no support for propagating modifications across related views or for manipulating visualization parameters by manipulating the views (with the exception of panning, zooming and rotating). The ability for Eikon to adjust the amount of detail that is displayed in an image can also be used to provide multiple small views of features from the data. Also as suggested by the case study participants Eikon should be extended to combine the glyph-based views with other visualization techniques such as contour plots, color images, and the ability to superimpose these images on glyph-based views. Other additions suggested included the ability to produce postscript images that could be seamlessly used in document production. The case study also indicated that the most useful practical addition would be the ability to generate movies automatically by incrementally varying the parameters associated with the visualization. In this manner animations showing the development of features in a time series can be produced.

Bibliography

[alpe91] B. Alpern and L. Carter. "The Hyperbox". *Visualization '91*, pp. 133–139, 1991.

[ande57] E. Anderson. "A Semigraphical Method for Analysis of Complex Problems". *Proceedings of the National Academy of Science*, Vol. 43, pp. 923–827, 1957.

[assa97] J. Assa, D. Cohen-Or, and T. Milo. "Displaying Data in Multidimensional Relevance Space with 2D Visualization Maps". *IEEE Visualization '97*, pp. 127–134, October 1997.

[avil94] R. Avila, T. He, L. Hong, A. Kaufman, H. Pfister, C. Silva, L. Sobierajski, and S. Wang. "VolVis: A Diversified Volume Visualization System". *IEEE Visualization '94*, pp. 31–38, October 1994.

[banc90] G.V. Bancroft, F.J. Merritt, T.C. Plessell, p.G. Kelaita, R.K. McCabe, and A. Globus. "FAST: A multi-Processed Environment for Visualization". *IEEE Visualization '90*, pp. 14–27, October 1990.

[bedd90] J. Beddow. "Shape Coding of Multidimensional Data on a Microcomputer Display". *IEEE Visualization '90*, pp. 238–246, October 1990.

[bede94] B. Bederson and J. Hollan. "Pad++: A Zooming Graphical Interface for Exploring Alternate". *UIST '94*, pp. 17–26, November 1994.

[berg95] L.D. Bergman, B.E. Rogowitz, and L.A. Treinish. "A Rule-based Tool for Assisting Colormap Selection". *IEEE Visualization '95*, pp. 118–125, 1995.

[bert83] J. Bertin. *Semiology of Graphics*. The University of Wisconsin Press, 1983.

[boyn79] R.M. Boynton. *Human Color Vision*. Holt Rinehart and Winston, 1979.

[bruc95] J.G. Bruce. "Eddies southwest of the Denmark Straigt". *Dee-Sea. Res.*, Vol. 45, pp. 13–29, 1995.

[brys91] S. Bryson and C. Levitt. "The Virtual Windtunnel: An Environment for the Exploration of Three-Dimensional Unsteady Flows". *Visualization '91*, pp. 17–24, 1991.

[buja91] Andreas Buja, John A. McDonald, John Michalak, and Werner Stuetzle. "Interactive Data Visualization Using Focusing and Linking". *Visualization '91*, pp. 156–163, 1991.

[busc99] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal. *A System of Patterns: Pattern - Oriented Software Architecture*. Wiley, 1999.

[cabr93] B. Cabral and L. Leedom. "Imaging Vector Fields Using Line Integral Convolution". *Computer Graphics (SIGGRAPH '93 Proceedings)*, Vol. 27, pp. 263–272, August 1993.

[cael78] T. Caelli, B. Julesz, and E. Gilbert. "On Perceptual Analyzers Underlying Visual Texture Discrimination: Part II". *Biological Cybernetics*, Vol. 29, pp. 201–214, 1978.

[carp95] M.S. Carpendale, D.J. Cowperthwaite, and F.D. Fracchia. "3-Dimensional Pliable Surfaces: For the Effective Presentation of Visual Information". *UIST '95*, pp. 217–226, November 1995.

[cher73] H. Chernoff. "The Use of Faces to Represent Points in k-Dimensional Space Graphically". *Journal of the American Statistical Asoociation*, Vol. 68, pp. 361–368, 1973.

[cher75] H. Chernoff and M.H. Rizvi. "Effect on classification error on random permutations of features in representing multivariate data by faces". *Journal of the American Statistical Asoociation*, Vol. 70, pp. 548–554, 1975.

[clev94] William S. Cleveland. *The Elements of Graphing Data.* Murray Hill, NJ, Bell Laboratories, 1994.

[clin88] H.E. Cline, W.E. Lorensen, S. Ludke, C.R. Crawford, and B.C. Teeter. "Two algorithms for the Three-Dimensional Construction of Tomograms". *Medical Physics*, Vol. 15, No. 3, pp. 320–327, June 1988.

[craw91] R.A. Crawfis and Allison M.J. "A Scientific Visualization Synthesizer". *Visualization '91*, pp. 262–267, 1991.

[desa84] G. DeSanctis. "Computer Graphics as Decision Aids: Directions for Research". *Decision Sciences*, Vol. 15, pp. 463–487, 1984.

[dyer90] D.S. Dyer. "A Dataflow Toolkit For Visualization". *IEEE Computer Graphics & Applications*, Vol. 10, No. 4, pp. 1990, July 1990.

[eber94] David Ebert, Kent Musgrave, Darwyn Peachey, Ken Perlin, and Worley. *Texturing and Modeling: A Procedural Approach.* Academic Press, October 1994.

[favr94] J.M. Favre and J. Hahn. "An Object-Oriented Design for Visualization of Multi-Variate Data Objects". *IEEE Visualization '94*, pp. 319–325, October 1994.

[ferr96] P. Ferry. "A Characterization of Flow Visualization Techniques". *Western Computer Graphics Symposium*, pp. 10–20, March 1996 1996.

[fole90] J.D. Foley, A. van Dam, S. K. Feiner, and J.F. Hughes. *Computer Graphics, Principles and Practice, Second Edition.* Addison-Wesley, Reading, Massachusetts, 1990.

[fors95] L.K. Forssell and S.D. Cohen. "Using Line Integral Convolution for Flow Visualization: Curvilinear Grids, Variable-Speed Anumation and Unsteady Flows". *IEEE Transactions on Visualization and Computer Graphics*, Vol. 1, No. 2, pp. 133–141, 1995.

[fren89] K.A. Frenkel. "Volume Rendering". *Communications of the ACM*, Vol. 4, pp. 426–435, April 1989.

[furn86] G.W. Furnas. "Generalized Fisheye Views". *Human Factors in Computing Systems (ACM SIGCHI'86 Conference Proceedings)*, pp. 18–23, 1986.

[furn95] G.W. Furnas and B.B. Bederson. "Space-Scale Diagrams: Uderstanding Multiscale Interfaces". *Human Factors in Computing Systems (ACM SIGCHI'95 Conference Proceedings)*, pp. 234–248, 1995.

[gall95] R.S. Gallagher, editor. *Computer Visualization.* CRC Press, 1995.

[garn81] W.R. Garner. "The Analysis of Unanalyzed Perceptions". *Perceptual Organization*, pp. 119–139, 1981.

[g.de86] G.DeSoete. "A Perceptual Study of the Flury-Riedwyl Faces for Graphically Displaying Multivariate Data". *International Journal of Man-Machine Studies*, Vol. 25, No. 5, pp. 549–555, 1986.

[glob91] A. Globus, C. Levit, and T. Lasinski. "A Tool for Visualizing the Topology of Three-Dimensional Vector Fields". *Visualization '91*, pp. 33–40, 1991.

[gonz93] R.C. Gonzalez and R.E. Woods. *Digital Image Processing.* Addison Wesley, 1993.

[grin89] G. Grinstein, R.M. Pickett, and M.G. Williams. "EXVIS: An exploratory visualization environment". *Proceedings of Graphics Interface '89*, pp. 254–261, June 1989.

[guit90] R. Guitard and C. Ware. "A Color Sequence Editor". *ACM Transactions on Graphics*, Vol. 9, No. 3, pp. 338–341, July 1990.

[habe90] R.B. Haber and D.A. McNabb. "Visualization Idioms: A Conceptual Model for Scientific Visualization Systems". *Visualization in Scientific Computing*, pp. 74–93, 1990.

[haid91] D.B. Haidvogel, A. Beckmand, and k.S. Hedstrom. "Dynamical simulations of filament formation and evolution in the coastal transition zone". *J. Geophys. Res.*, Vol. 96, pp. 15,071–15,040, 1991.

[hain98] T.W.N. Haines, A.J. Watson, M.I. Liddicoat, and R.R. Dickson. "The flow of Antarctic bottom water to the southwest Indian Ocean estimated using CFCs". *J. Geophys. Res.*, Vol. 103, No. 27, pp. 637–653, 1998.

[hait80] M.M. Haith. *Rules that babies look by: The organization of new born visual activity.* Lawerence Erlbaum Associates, 1980.

[hall86] E. Halley. "An Historical Account of the Trade Winds, and Monsoons, Observable in the Seas Between and Near the Tropicks; With an Attempt to Assign the Phisical Cause of Said Winds". *Philosophical Transactions,* Vol. 183, pp. 153–168, 1686.

[hara73] R. M. Haralick, K. Shanmugam, and I. Dinstein. "Textural Features for Image Classification". *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. 3, No. 6, pp. 610–621, november 1973.

[hart75] J.A. Hartigan. "Printer Graphics for Clustering". *Journal of Statistical Computation and Simulation*, Vol. 4, pp. 187–213, 1975.

[heal98] C.G. Healey and J.T. Enns. "Building Perceptual Textures to Visualize Multidimension Datasets". *Visualization '98*, pp. 111–118, 1998.

[heeg95] David J. Heeger and James R. Bergen. "Pyramid-Based Texture Analysis/Synthesis". *SIGGRAPH 95 Conference Proceedings*, Annual Conference Series, pp. 229–238, August 1995.

[helm90a] J. Helman and L. Hesselink. "Representation and display of vector field topology in fluid flow data sets". *Visualization in scientific computing*, pp. 61–73, 1990.

[helm90b] J. L. Helman and L. Hesselink. "Surface Representations of Two- and Three- Dimensional Fluid Flow Topology". *IEEE Visualization '90*, pp. 6–13, October 1990.

[helm91] J. L. Helman and L. Hesselink. "Visualization of Vector Field Topology in Fluid Flows". *IEEE Computer Graphics and Applications*, Vol. 11, No. 3, pp. 36–46, 1991.

[hess97] L. Hesselink, Y. Levy, and Y. Lavin. "The Topology of Symmetric, Second-Order 3D Tensor Fields". *IEEE Transactions on Visualization and Computer Graphics*, Vol. 3, No. 1, January–March 1997.

[hibb90] B. Hibbard and D. Santek. "The VIS-5D System for Easy Interactive Visualization". *IEEE Visualization '90*, pp. 28–35, October 1990.

[hugh96] R.G. Hughes and A.R. Forrest. "Perceptualisation using a Tactile Mouse". *IEEE Visualization '96*, pp. 181–188, October 1996.

[hult92] J.P.M. Hultquist. "Constructing Stream Surfaces in Steady 3D Vector Fields". *IEEE Visualization '92*, pp. 171–178, October 1992.

[IBM91] IBM Corp., Armonk, NY. *Data Explorerer Reference Manual*, 1991.

[jog94] N. Jog and B. Shneiderman. "Starfield Information Visualization with Interactive Smooth Zooming". Technical Report Technical Report CS-TR-3286, University of Maryland, 1994.

[jule73] B. Julesz, E.N. Gilbert, and L.A. Shepp. "Inability of humans to discriminate between visual textures that agree in secon-order statistics – revisited". *Perception*, Vol. 2, pp. 391–405, 1973.

[jule81] B. Julesz. "Textons, the elements of texture perception, and their interactions". *Nature*, Vol. 290, No. 12, pp. 91–97, March 1981.

[kauf90] A. Kaufman. *Volume Visualization*. IEEE Computer Society Press, 1990.

[keim94a] D.A. Keim and Hans-Petter Kriegel. "VisDB: Database Exploration Using Multimimensional Visualization". *IEEE Computer Graphics and Applications*, pp. 40–49, September 1994.

[keim94b] D.A. Keim and Hans-Petter Kriegel. “Visualization Techniques for Mining Large Databases: A Comparison”. *IEEE Transactions on Knowledge and Data Engeering*, Vol. 8, No. 6, pp. 1–23, December 1994.

[keim96] D.A. Keim. “Pixel-oriented Database Visualizations”. *SIGMOD RECORD special issue on 'Information Visualization'*, December 1996.

[kerl90] G.D. Kerlick. “Moving Iconic Objects in Scientific Visualization”. *Visualization '90*, pp. 124–129, October 1990.

[kiu96] M.H. Kiu and D.C. Banks. “Multi-Frequency Noise for LIC”. *IEEE Visualization '96*, October 1996.

[klas91] R.V. Klassen and S.J. Harrington. “Shadowed Hedgehogs: A Technique for Visualizing 2D Slices of 3D Vector Fields”. *Visualization '91*, pp. 148–153, 1991.

[klei81] B. Kleiner and J.A. Hartigan. “Representing Points in Many Dimensions by Trees and Castles”. *Journal of the American Statistical Association*, Vol. 76, pp. 260–269, June 1981.

[kons94] K. Konstantinides and J.R. Rasure. “The Khoros software development environment for image and signal processing”. *IEE Transactions on Image Processing*, Vol. 3, No. 3, pp. 243–252, 1994.

[laid98] D. Laidlaw, E.T. Ahrens, D. Kremers, J. Avalos, C. Readhead, and R. Jacobs. “Visualizing Diffusion Tensor Images of the Mouse Spinal Cord”. *Visualization '98*, pp. 137–134, 1998.

[laid99] D. Laidlaw, R. Kirby, and H. Harmanis. “Visualizing Multivalued Data from 2D Incompressible flows using concepts from painting”. *Visualization '99*, pp. 333–340, 1999.

[lamb79] J. H. Lambert. *Pyrometrie.* Berlin, 1779.

[l'ec32] Commision Internationale de l'Eclairage. “Proceedings 1931”, 1932.

[l'ec71] Commision Internationale de l'Eclairage. "CIE Publication No. 15, Supplement Number 2(E-1.3.1): Official Recommendations on Uniform Color Spaces, Color-Difference Equations and Metric Color Terms", 1971.

[leeu93] W.C. de Leeuw and J.J. van Wijk. "A Probe for Local Flow Field Visualization". *IEEE Visualization '93*, pp. 39–45, October 1993.

[leeu95] W.C. de Leeuw and J. van Wijk. "Enhance Spot Noise for Vector Field Visualization". *IEEE Visualization '95*, pp. 233–239, 1995.

[lege91] S.M. Legensky. "Advanced Visualization on Desktop Workstations". *Visualization '91*, pp. 372–378, 1991.

[levk91] H. Levkowitz. "Color Icons: Merging Color and Texture Perception for Integrated Visualization of Multiple Parameters". *Visualization '91*, pp. 164–170, 1991.

[luca92] B. Lucas, G. Abram, N. Collins, D. Epstein, D. Gresh, and K. McAuliffe. "An Architecture for a scientific visualization system". *IEEE Visualization '92*, pp. 107–114, October 1992.

[maca70] D. L. MacAdam. *Sources of Color Science.* M.I.T Press, Cambridge, MA, 1970.

[mack91] J. Mackinlay, G. Robertson, and S. Card. "The Persepective Wall: Detail and Context Smoothly Integrated". *Human Factors in Computing Systems (ACM SIGCHI'91 Conference Proceedings)*, pp. 173–179, 1991.

[max93] N. Max, B. Becker, and R. Crawfis. "Flow Volues for Interactive Vector Field Visualization". *IEEE Visualization '93*, pp. 19–23, October 1993.

[mcco87] B. McCormick. "Visualization in scientific computing". *Computer Graphics*, Vol. 21, No. 6, 1987.

[meye80] G.W. Meyer and D.P. Greenberg. "Perceptual Color Spaces for Computer Graphics". *Computer Graphics (SIGGRAPH '80 Proceedings)*, Vol. 14, No. 3, pp. 254–261, July 1980.

[ming95] R. Minghim and R. Forrest. "An Illustrated Analysis of Sonification for Scientific Visualization". *IEEE Visualization '95*, pp. 110–117, 1995.

[muld95] J.D. Mulder and J. van Wijk. "3D Computational Steering with Parametrized Geometric Objects". *IEEE Visualization '95*, pp. 304–312, 1995.

[muns94] H. Munsterberg. *Pseudoptics.* New York: Milton Bradley, 1894.

[niel90] G.M. Nielson and B.D. Shriver (editors). *Visualization in Scientific Computing.* IEEE Computer Society, 1990.

[pang93] A. Pang and K. Smith. "Spray Rendering: Visualization using smart particles". *IEEE Visualization '93*, pp. 283–290, October 1993.

[pang94] A. Pang and N. Alper. "Mix&Match: A Construction Kit for Visualization". *IEEE Visualization '94*, pp. 302–318, October 1994.

[pang97] A. Pang and C. Wittenbrink. "Collaborative 3D Visualization with CSpray". *IEEE Computer Graphics & Applications*, Vol. 17, No. 2, pp. 32–41, March-April 1997.

[perl85] Ken Perlin. "An Image Synthesizer". *Computer Graphics (SIGGRAPH '85 Proceedings)*, Vol. 19, pp. 287–296, July 1985.

[perl93] K. Perlin and D. Fox. "An Alternative Approach to the Computer Interface". *Computer Graphics (SIGGRAPH '93 Proceedings)*, Vol. 27, pp. 57–64, August 1993.

[perl95] Ken Perlin and Luiz Velho. "Live Paint: Painting With Procedural Multiscale Textures". *SIGGRAPH 95 Conference Proceedings*, Annual Conference Series, pp. 153–160, August 1995.

[pham90] B. Pham. "Spline-based Color Sequences for Univariate, Bivariate and Trivariated Mapping". *IEEE Visualization '90*, pp. 202–208, October 1990.

[phil88] D. Phillips. "Siggraph '88 Panels Proceedings", 1988.

[pick66] R.M. Pickett and B.W. White. "Constructing Data Pictures". *Information Display*, Vol. 3, No. 5, pp. 57, 1966.

[pick88] R. Pickett and G. Grinstein. "Iconographic displays for visualizing multidimensional data". *Proceedings of the 1988 IEEE Conference on Systems, Man, and Cybernetics*, pp. 514–519, 1988.

[play86] W. Playfair. *The Comperical and Political Atlas*. London:Corry, 1786.

[post95] F.J. Post, T. van Walsum, F.H. Post, and D. Silver. "Iconic Techniques for Feature Visualization". *IEEE Visualization '95*, pp. 288–295, 1995.

[rao93] A.R. Rao and G.L. Lohse. "Towards a Texture Naming System: Identifying Relevant Dimenisons of Texture". *IEEE Visualization '93*, pp. 220–227, October 1993.

[reev83] W. T. Reeves. "Particle Systems – a Technique for Modeling a Class of Fuzzy Objects". *ACM Trans. Graphics*, Vol. 2, pp. 91–108, April 1983.

[resz97] M.K. Reszka. "Finite amplitude waves and eddy formation on a baroclinic unstable front over a sloping bottom". M.Sc. Thesis, Department of Mathematical Sciences, University of British Alberta, 1997.

[resz99] M.K. Reszka and G.E. Swaters. "Eddy formation and interaction in a baroclinic frontal geostrophic model". *J. Phys. Oceanogr.*, Vol. 29, pp. 3025–3042, 1999.

[rhei90] P. Rheingans and B. Tebbs. "A Tool for Dynamic Explorations of Color Mappings". *Computer Graphics (1990 Symposium on Interactive 3D Graphics)*, Vol. 24, No. 2, pp. 145–146, March 1990.

[rhei92] P. Rheingans. "Color, Change and Control for Quantitative Data Display". *IEEE Visualization '92*, pp. 252–259, October 1992.

[rhei97] P. Rheingans. "Dynamic Color Mapping of Bivariate Qualitative Data". *IEEE Visualization '97*, pp. 159–166, October 1997.

[riba94] W. Ribarsky, E. Ayers, J. Eble, and S. Mukherjea. "Glyphmaker: Creating Customized Visualizations of Complex Data". *Computer*, Vol. 27, No. 7, pp. 57–64, July 1994.

[rich90] R. Richter, J.B. Vos, A. Bottaro, and S. Gavrilakis. "Visualization of Flow Simulations". *Scientific Visualization and Graphics Simulations*, pp. 161–171, 1990.

[robe86] P.K. Robertson and J.F. O'Callaghan. "The Generation of Color Sequences for Univariate and Bivariate Mapping". *IEEE Computer Graphics and Applications*, Vol. 6, No. 2, pp. 24–32, February 1986.

[robe88] P.K. Robertson. "Visualizing color gamuts: a user interface for the effective use of perceptual color spaces in data displays". *IEEE Computer Graphics and Applications*, Vol. 8, No. 5, pp. 50–64, September 1988.

[robe90] P.K. Robertson. "A Methodology for Scientific Data Visualization: Choosing Representations Based on a Natureal Scene Paradigm". *IEEE Visualization '90*, pp. 114–123, October 1990.

[robe93] G.G. Robertson and J.D. Mackinlay. "The Document Lens". *UIST '93*, pp. 101–108, November 1993.

[rogo93a] B.E. Rogowitz and L.A. Treinish. "An Architecture for Rule-Based Visualization". *IEEE Visualization '93*, pp. 236–243, October 1993.

[rogo93b] B.E. Rogowitz and L.A. Treinish. "Data Structures and Perceptual Structures". *Proceedings of the SPIE/SPSE Spymposium on Electronic Imaging*, 1993.

[sanc00] J. Sanchez and M.P. Canton. *DirectX 3D Graphics Programming Bible*. Hungry Minds, 2000.

[sark92] M. Sarkar and M. Brown. "Graphical Fisheye Views of Graphs". *Human Factors in Computing Systems (ACM SIGCHI'92 Conference Proceedings)*, pp. 83–91, 1992.

[sark93] M. Sarkar, S.S. Snibe, O.J. Tversky, and S.P. Reiss. "Stretching the Rubber Sheet: A Metaphor for Viewing Large Layouts on Small Screens". *UIST '93*, pp. 81–91, November 1993.

[schr91] W. Schroeder, C.R. Volpe, and W.E. Lorensen. "The Stream Polygon: A Technique for 3D Vector Field Visualization". *Visualization '91*, pp. 126–132, 1991.

[schr98] W. Schroeder, K. Martin, and B. Lorensen. *The Visualization Toolkit: An Object-Oriented Approach to 3D Graphics*. Prentice Hall, 1998.

[shaw98] C.D. Shaw, D.S. Ebert, J.M. Kukla, A. Zwa amd O. Soboroff, and D.A. Roberts. "Data Visualization Using Automatic Perceptually-Motivated Shapes". *Proceedings of the SPIE/SPSE Symposium on Electronic Imaging*, 1998.

[sieg71] J.H. Siegel, R.M. Goldwyn, and H.P. Friedman. "Pattern and Process in the Evolution of Human Septic Shock". *Surgery*, Vol. 70, No. 2, pp. 232–245, August 1971.

[Sil91] Silicon Graphics Inc., Mountain View, CA. *Iris Explorer User's Guide*, 1991.

[smit78] A.R. Smith. "Color gamut transform pairs". *Computer Graphics (SIGGRAPH '78 Proceedings)*, Vol. 12, No. 3, pp. 12–19, August 1978.

[smit91] S. Smith, G. Grinstein, and R. Bergeron. "Interactive Data Exploration with a Supercomputer". *Visualization '91*, pp. 248–254, 1991.

[stac87] J.W. Stacey, S. Pond, P.H. LeBlond, H.J. Freeland, and D.M. Farmer. "An analysis of the low-frequency fluctuations in the Strait of Georgia from June 1984 to January 1985". *J. Phys. Oceanogr.*, Vol. 17, pp. 326–342, 1987.

[stal95] D. Stalling and H.C. Hege. "Fast and Resolution Independent Line Integral Convolution". *IEEE Visualization '95*, pp. 249–256, 1995.

[stew99] J. Stewarts. *Calculus 4th Edition.* Brookes/Cole, 1999.

[stol91] J. Stolk and J.J. van Wijk. "Surface Particles for 3D Flow Visualization". *Proceedings of the Second Eurographics Workshop on Visualization in Scientific Computing*, pp. 22–24, 1991.

[stro98] T. Strothotte. *Computational Visualization: Graphics, Abstraction, and Interactivity.* Springer, 1998.

[taji83] J. Tajima. "Uniform Color Scale Applications to Computer Graphics". *Computer Vision, Graphics and Image Processing*, Vol. 21, No. 3, pp. 305–325, March 1983.

[tamu78] H. Tamura, S. Mori, and T. Yamawaki. "Textural features corresponding to visual percetion". *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. 8, No. 6, pp. 460–473, june 1978.

[trum81] B.E.A Trumbo. "Theory for Coloring Bivariate Statistical Maps". *The American Statistician*, Vol. 35, No. 4, pp. 220–226, 1981.

[tuft83] E.R. Tufte. *The Visual Display of Quantitative Information.* Graphics Press, Cheshire, Connecticut, 1983.

[tuft91] E.R. Tufte. *Envisioning Information.* Graphics Press, Cheshire, Connecticut, 1991.

[tuft97] E.R. Tufte. *Visual Explanations.* Graphics Press, Cheshire, Connecticut, 1997.

[upso89] C. Upson. "The Application Visualization System: A computational environment for scientific visualization.". *IEEE Computear Graphics and Applications*, Vol. 9, No. 4, pp. 30–42, 1989.

[volp97] C.R. Volpe and E.P. Glinert. "Auralization of Streamline Vorticity in Computational Fluid Dynamics Data". *IEEE Visualization '97*, pp. 51–57, October 1997.

[wain80] H. Wainer and C.M. Francolini. "An Empirical Enquiry Concerning Human Understanding of Two Variable Color Maps". *The American Statistician*, Vol. 34, No. 2, pp. 81–93, 1980.

[ware88a] Colin Ware. "Color sequences for univariate maps: theory, experiments, and principles". *IEEE Computer Graphics and Applications*, Vol. 8, No. 5, pp. 41–49, September 1988.

[ware88b] Colin Ware and John Beatty. "Using Color Dimensions to Display Data Dimensions". *Human Factors*, Vol. 30, No. 2, pp. 127–142, 1988.

[ware95] C. Ware and W. Knight. "Using Visual Texture for Information Display". *ACM Transactions on Graphics*, Vol. 14, No. 1, pp. 3–20, January 1995.

[Wav90] Waverfront Technologies, Santa Barbara, CA. *Data Visualizer User Manual*, 1990.

[wege97] R. Wegenkittl and E.Groller. "Fast Oriented Line Integral Convolution for Vector Field Visalization via the Internet". *IEEE Visualization '97*, pp. 309–316, October 1997.

[wijk91] J.J. van Wijk. "Spot noise-Texture Synthesis for Data Visualization". *Computer Graphics (SIGGRAPH '91 Proceedings)*, Vol. 25, pp. 309–318, July 1991.

[wijk92] J.J. van Wijk. "Rendering Surface Particles". *IEEE Visualization '92*, pp. 54–61, October 1992.

[wijk93] J.J. van Wijk and R. van Liere. "HyperSlice – Visualization of Scalar Functions of Many Variables". *IEEE Visualization '93*, pp. 119–225, 1993.

[wilk00] L. Wilkinson. *The Grammar of Graphics.* Springer, 2000.

[will83] Lance Williams. "Pyramidal Parametrics". *Computer Graphics (SIGGRAPH '83 Proceedings)*, Vol. 17, pp. 1–11, July 1983.

[witt96] C.M. Wittenbrink, A. T. Pang, and S. K. Lodha. "Glyphs for Visualizing Uncertainty in Vector Fields". *IEEE Transactions on Visualization and Computer Graphics*, Vol. 2, No. 3, pp. 266–279, September 1996.

[woo99] M. Woo, J. Neider, and T. Davisand D. Shreiner. *OpenGL Programming Guide*. Addison-Wesley Publishing Company, 1999.

[wysz82] G. Wyszecki and W. S. Stiles. *Color Science: Concepts and Methods, Quantitative Data and Formulae.* Wiley, 1982.

[youn92] M.P. Young and S. Yamane. "Sparse Population Coding of Faces in Inferotemporal Cortex". *Science*, Vol. 56, pp. 1326–1331, 1992.